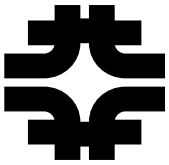

GU0014B

Complete Guide and Reference Manual for UPS, UPD and UPP v4

Part II: Product Installer's Guide
Part III: System Administrator's Guide
Part V: Distribution Node Maintainer's Guide and
Part VII: Administrator's Reference

Release 2.0
June 30, 2000

Computing Division
Fermi National Accelerator Laboratory

Compiled by Anne Heavey

ABSTRACT

This manual documents the standard methodology for UNIX product support at Fermilab, which is implemented via the utilities **UPS** (UNIX Product Support), **UPD** (UNIX Product Distribution), and **UPP** (UNIX Product Poll). These utilities were significantly redesigned for version v4, which was initially released in 1998, and have continued to be revised since then. The latest release as of this writing is v4_5_2. This document supersedes GU0014 "UPS and UPD v4 Reference Manual", released June 5, 1998.

This part of the document (GU0014B) includes separate user's guides for product installers, **UPS/UPD** and system administrators, and maintainers of product distribution nodes. It also includes a reference guide for administrative users.

Revision Record

May 1997	Original Release 1.0 (for UPS v3 and UPD v2)
August 1997	Revisions 1.1 and 1.1a (for UPS v3 and UPD v2)
June 1998	Release 1.0 for UPS and UPD v4
December 1999	Draft release 2.0 for UPS/UPD/UPP v4. Part VI Command Reference only
June 2000	Release 2.0 for UPS, UPD and UPP v4 (current as of v4_5_2)

Acknowledgments

The redesign and redevelopment of **UPS** and its companion products in preparation for Fermilab's Run II involved a substantial commitment of resources from the Computing Division in 1997-98. Special thanks to Don Petravick (HPPC), Ruth Pordes (OLS), and Dane Skow (OSS) for providing talented and motivated members of their groups to accomplish this task. Since the initial release of **UPS/UPD v4** in 1998, development has been continuing, and we are at version v4_5_2 as of this writing.

The redevelopment effort was led by Eileen Berman. With her, the principal designers and developers of **UPS/UPD** v4 included David Fagan, Marc Mengel, Lars Rasmussen and Margaret Votava. Other contributors to the new design included Lauri Loebel Carpenter, Rob Harris, Alan Jonckheere, Art Kreymer, Liz Sexton-Kennedy. Other contributors to the coding effort included Chuck Debaun, Paul Russo and Don Walsh.

Contributors in the areas of code review, testing, documentation review and deployment included Lauri Loebel Carpenter, Chuck Debaun, Lisa Giacchetti, Alan Jonckheere, Art Kreymer, Liz Sexton-Kennedy, Mike Stolz, Don Walsh and Gordon Watts, in addition to the development team. Special thanks go to Marc Mengel and Margaret Votava for contributing all the updated **UPD** and **UPP** information included in the first release of this manual for **UPS/UPD v4**.

Wayne Baisley and Marc Mengel are currently responsible for on-going support and development of **UPS/UPD**, and thanks go to them for providing quite a bit of updated information for this release of the manual. Thanks are also due to Wayne and Marc as well as to Joy Hathaway, Lauri Loebel Carpenter and Cindy Wike for reviewing portions of the documentation and providing feedback.

Table of Contents for Parts II, III, V, and VII

About this Manual INT-1
Document Structure, Purpose and Intended Audiences. INT-1
Availability INT-3
Updates. INT-3
Conventions INT-3
Your Comments are Welcome! INT-5
Glossary GLO-1
Index IDX-1

Part II: Product Installer's Guide

Chapter 3: General Product Installation Information 3-1
3.1 Installation Methods for UPS Products 3-1
3.1.1 UPD 3-1
3.1.2 UPP 3-2
3.1.3 FTP 3-2
3.2 User Node Registration for KITS 3-2
3.3 What You Need to Know about Your System's UPD Configuration . . 3-3
3.3.1 Location of UPD Configuration File 3-3
3.3.2 Where Products Get Declared 3-4
3.3.3 Where Products Get Installed 3-4
3.4 Declaring an Instance Manually 3-5
3.4.1 The ups declare Command 3-5
3.4.2 Examples 3-6
3.5 Installation FAQ 3-7
3.5.1 What File Permissions Get Set? 3-7
3.5.2 You're Ready to Install: Should you Declare Qualifiers? 3-8
3.5.3 What if an Install Gets Interrupted? 3-8
3.5.4 What if a Product was Installed under a Different Name? 3-8
3.6 Post-Installation Procedures 3-9
3.6.1 Configuring a Product 3-9
3.6.2 Tailoring a Product 3-9

3.7 Networking Restrictions at your Site 3-9
3.7.1 Proxying Webserver 3-9
3.7.2 Firewall for Incoming TCP Connections 3-10
Chapter 4: Finding Information about Products on a Distribution Node . 4-1
4.1 Listing Products on a Distribution Node 4-1
4.1.1 Using UPD 4-1
4.1.2 Using UPP 4-3
4.2 Listing Product Dependencies on a Distribution Node 4-5
4.3 Information about Products in KITS 4-6
4.3.1 Access Restrictions and Product Categories 4-6
4.3.2 Product Pathnames for FTP Access 4-7
4.4 Special Instructions for Proprietary Products 4-8
Chapter 5: Installing Products Using UPD 5-1
5.1 The upd install Command 5-1
5.1.1 Command Syntax 5-1
5.1.2 Passing Options to the Local ups declare Command 5-2
5.2 How UPD Selects the Database 5-2
5.2.1 Database Selection Algorithm 5-2
5.2.2 Database Selection for Dependencies 5-3
5.2.3 Selecting a Database for Development or Testing 5-3
5.3 Checklist for Installing a Product using UPD 5-3
5.4 Examples 5-4
5.4.1 Install a Product Using Default Database 5-4
5.4.2 Install a Product, Specifying Database 5-5
5.4.3 Install a Product and All Dependencies 5-5
5.4.4 Install a Product and No Dependencies 5-7
5.4.5 Install a Product and Required Dependencies Only 5-7
Chapter 6: Installing Products Using UPP 6-1
6.1 Overview of Using UPP to Install Products 6-1
6.2 Creating a UPP Subscription File 6-1
6.2.1 Create the Header 6-2
6.2.2 Identify the Product 6-2
6.2.3 Trigger the Product Installation 6-2
6.2.4 Provide Instructions to UPP 6-3
6.3 Sample Subscription File for Installing a Product 6-3
6.4 The UPP Command 6-4
6.5 Automating UPP via cron 6-4
Chapter 7: Installing Products using FTP 7-1
7.1 UPS Product Components to Download 7-1
7.2 Installing Products from fnkits.fnal.gov 7-2
7.2.1 Download the Files from fnkits 7-2
7.2.2 Unwind the Files into your Products Area 7-3
7.2.3 Declare the Product to your Database 7-4
7.3 Installing Products from Other Product Distribution Nodes 7-4

7.3.1 Locate the Product Files on the Server 7-4
7.3.2 Download the Files from the Server 7-5
7.3.3 Unwind the Files into your Products Area 7-5
7.3.4 Declare the Product to your Database 7-5

Chapter 8: Product Installation: Special Cases 8-1

8.1 Installing Products that Require Special Privileges 8-1
8.2 Installing Locally Using UPD from AFS-Space 8-2
8.3 Installing Products into AFS Space 8-3
8.3.1 Overview 8-3
8.3.2 Request a Product Volume 8-4
8.3.3 Install your Product 8-4
8.3.4 Post-Installation Steps 8-5

Chapter 9: Troubleshooting UPS Product Installations 9-1

Part III: System Administrator's Guide

Chapter 10: Maintaining a UPS Database 10-1

10.1 Declare an Instance 10-1
10.1.1 The ups declare Command 10-2
10.1.2 Examples 10-2
10.2 Declare a Chain 10-4
10.2.1 The ups declare Command with Chain Specification 10-4
10.2.2 Examples 10-5
10.3 Remove a Chain 10-6
10.4 Change a Chain 10-7
10.5 Undeclare and Remove an Instance 10-7
10.5.1 Using ups undeclare to Remove a Product 10-8
10.5.2 Undoing Configuration Steps 10-9
10.5.3 Using UPP to Remove a Product 10-10
10.6 Verify Integrity of an Instance 10-10
10.7 Modify Information in a Database File 10-11
10.8 Determine If a Product Needs to be Updated 10-13
10.8.1 Using UPP 10-13
10.8.2 Using UPD 10-13
10.9 Update a Table File or ups Directory 10-14
10.10 Retrieve an Individual File 10-15
10.11 Check Product Accessibility 10-16
10.12 Troubleshooting 10-17

Chapter 11: UPS and UPD Pre-install Issues and General Administration
. 11-1
11.1 Choosing Installer Accounts 11-1
11.1.1 Single Installer Account 11-1
11.1.2 Multiple Installer Accounts 11-1
11.1.3 Separate Installer Accounts for Different Product Categories . . 11-2
11.2 Setting gids for Multiple Installer Accounts 11-2
11.3 File Ownership, Permissions and Access Restrictions 11-3
11.3.1 Product Files 11-3
11.3.2 Database Files 11-3
11.4 Product File Location and Organization 11-4
11.4.1 Considerations 11-4
11.4.2 Single Flavor or Single Node Systems 11-4
11.4.3 Multi-Flavor and/or Multi-Node Systems 11-5
11.5 Database File Location and Organization 11-6
11.5.1 Choosing Single or Multiple UPS Databases 11-6
11.5.2 UPS Database File Pointers 11-6
11.6 Installing UPS for Use Without a Database 11-7
11.7 CYGWIN (Windows NT) Issues 11-7
11.7.1 Using Correct Perl Version 11-7
11.7.2 Mounting the CYGWIN bin Directory 11-8
11.7.3 Setting Environment Variables 11-8
11.8 General Administration Issues 11-8
11.8.1 Upgrading an Older System 11-8
11.8.2 Adding a New Database and/or Products Area 11-9
11.8.3 Collecting Statistics on Product Usage 11-10
Chapter 12: Providing Access to AFS Products 12-1
12.1 Overview 12-1
12.2 Configuring a Local Database to Work With AFS 12-2
12.2.1 Steps to Create and Configure the Database 12-2
12.2.2 Post-Configuration: Reinitialize FUE Environment 12-4
12.2.3 A Note about Product Installation for this Configuration 12-4
12.3 Installing a Local Copy of CoreFUE 12-4
12.4 Additional Steps for Unfamiliar Naming Conventions 12-5
12.5 Updating /usr/local/bin to Access AFS Products 12-6
Chapter 13: Bootstrapping CoreFUE 13-1
13.1 Downloading the Bootstrap and Configuration Files 13-1
13.1.1 Predefined Configurations for UNIX 13-1
13.1.2 User-defined Configuration for UNIX 13-2
13.1.3 Predefined Configurations for NT 13-2
13.2 Customizing a Bootstrap Configuration 13-3
13.2.1 Bootstrap Configuration File Statement Definitions 13-3
13.2.2 Sample Customization 13-4

13.3 Running the Bootstrap Procedure 13-5
13.3.1 UNIX 13-5
13.3.2 NT 13-5
Chapter 14: Automatic UPS Product Startup and Shutdown 14-1
14.1 Configuring Your Machine to Allow Automatic Startup/Shutdown . 14-1
14.2 Installing a UPS Product to Start and/or Stop Automatically 14-2
14.2.1 Determine if Auto Start/Stop Feature is Enabled 14-2
14.2.2 Determine if Product is Appropriate for Autostart 14-3
14.2.3 Edit Control File(s) 14-3
14.2.4 Summary 14-4
14.3 Disabling UPS Automatic Start/Stop of Processes 14-4
14.4 A Summary of the UPS Automatic Start-up Process 14-5

Part V: Distribution Node Maintainer's Guide

Chapter 20: Product Distribution Server Configuration 20-1
20.1 How A Server Responds to a UPD Client Command 20-1
20.1.1 The Process for upd addproduct 20-2
20.1.2 The Process for upd install 20-2
20.2 Accounts Required for Distribution Server 20-3
20.2.1 The updadmin Account 20-3
20.2.2 The ftp Account 20-3
20.2.3 The wwwadm Account 20-4
20.3 Web Server Configuration 20-5
20.3.1 The cgi Scripts Used to Access Distribution Database 20-5
20.3.2 Restricting Access to Distribution Database 20-6
20.3.3 Prerequisites for Modifying the Distribution Database 20-7
20.3.4 Permissions on Files Created in the Distribution Database 20-7
20.4 FTP Server Configuration 20-7
20.5 UPD Configuration Items 20-9
20.5.1 Archive File Keywords and ${SUFFIX} 20-9
20.5.2 Pre- and Postdeclare ACTIONs 20-10
20.6 Administrative Tasks and Utilities 20-10
20.6.1 Reporting FTP and Web Server Activity Using Ftpweblog . . 20-10
20.6.2 Restricting Access for Uploads to Distribution Database 20-11
20.6.3 Restricting Access for Downloads from Distribution Database 20-11
20.6.4 Restricting Distribution of Particular Products 20-11
20.6.5 Flagging Special Category Products Using Optionlist 20-12
20.6.6 Searching FTP Server Logfiles Using Searchlog 20-13
20.7 Product Distribution via CD-ROM 20-14
Chapter 21: Configuration of the fnkits Product Distribution Node 21-1
21.1 UPS Configuration for KITS Database 21-1
21.2 UPS Configuration for local Product Database 21-1

21.3 UPD Configuration 21-2
21.3.1 updconfig File Organization 21-2
21.3.2 The Recognized Product Categories 21-3
21.3.3 Matching Product Categories to updconfig Stanzas 21-3
21.3.4 Location and File Name Definitions 21-4
21.3.5 Pre- and Postdeclare ACTIONS 21-4
21.4 fnkits Server Maintenance 21-6
21.4.1 User Accounts and Group Ids 21-6
21.4.2 Database and Configuration File Locations 21-6
21.4.3 Web Server and FTP Log File Information 21-7

Part VII: Administrator's Reference

Chapter 27: Information Storage Format in Database and Configuration Files 27-1
27.1 Overview of File Types 27-1
27.2 Keywords: Information Storage Format 27-2
27.2.1 What is a Keyword? 27-2
27.2.2 Keyword Syntax 27-2
27.2.3 User-Defined Keywords 27-2
27.2.4 How UPS/UPD Sets Keyword Values 27-3
27.3 Flexibility of File Syntax 27-3
27.4 List of Supported Keywords 27-3
27.5 Syntax for Assigning Keyword Values 27-8
27.6 Usage Notes on Particular Keywords 27-9
27.6.1 COMPILE_DIR, COMPILE_FILE and @COMPILE_FILE . . 27-9
27.6.2 PROD_DIR_PREFIX, PROD_DIR and @PROD_DIR 27-9
27.6.3 STATISTICS 27-9
27.6.4 TABLE_FILE and @TABLE_FILE 27-10
27.6.5 UPS_DIR and @UPS_DIR 27-11
27.6.6 _UPD_OVERLAY 27-11
Chapter 28: Version Files 28-1
28.1 About Version Files 28-1
28.2 Keywords used in Version Files 28-2
28.3 Version File Examples 28-3
28.3.1 Sample Version File for exmh v1_6_6 28-3
28.3.2 Sample version file for foo v2_0 28-4
28.4 Determination of ups Directory and Table File Locations 28-5
Chapter 29: Chain Files 29-1
29.1 About Chain Files 29-1
29.2 Keywords Used in Chain Files 29-2

29.3 Chain File Examples 29-3
29.3.1 Sample chain file for exmh v1_6_6 29-3
29.3.2 Sample chain file for foo v2_0 29-3
Chapter 30: The UPS Configuration File 30-1
30.1 dbconfig File Organization 30-1
30.2 Keywords Used in dbconfig 30-1
30.3 Sample dbconfig File 30-2
Chapter 31: The UPD Configuration File 31-1
31.1 updconfig File Organization 31-1
31.2 Product Instance Identification and Matching 31-2
31.3 Defining Locations for Product Files 31-3
31.3.1 Required Locations 31-3
31.3.2 Read-Only Variables Usable in Location Definitions 31-4
31.3.3 Sample Location Definitions 31-5
31.4 Pre- and Postdeclare Actions 31-5
31.4.1 ACTION Keyword Values 31-6
31.4.2 The execute Function 31-6
31.5 Examples 31-7
31.5.1 Generic Template updconfig File 31-7
31.5.2 Distribution from the fnkits Node Only 31-8
31.5.3 Customized Treatment of ups Directory and Table Files 31-8
31.5.4 Implementing Multiple Configurations 31-9
31.5.5 Sample Configuration for AFS Space Using ACTIONS 31-10
31.5.6 Distribution Node Configuration 31-10
Chapter 32: The UPP Subscription File 32-1
32.1 UPP Subscription File Header 32-1
32.2 Stanzas 32-2
32.2.1 Product Instance Identification 32-2
32.2.2 Conditions and Instructions 32-2
32.3 Examples 32-3
32.3.1 Sample UPP Subscription File 32-3
32.3.2 A Longer Annotated Example 32-4
Glossary GLO-1
Index IDX-1

Table of Contents for Complete Guide

About this Manual INT-1

(This introductory chapter is listed in the front section of the table of contents.)

Part I: Overview and End User's Guide

Chapter 1: Overview of UPS, UPD and UPP v4 1-1

1.1 Introduction to UPS, UPD and UPP 1-1
1.2 Motivation for the UPS Methodology 1-2
1.3 UPS Products 1-3
1.3.1 Versions 1-3
1.3.2 Flavors 1-3
1.3.3 Qualifiers 1-4
1.3.4 Product Instances 1-4
1.3.5 Chains 1-4
1.3.6 Product Dependencies 1-5
1.3.7 Product Overlays 1-6
1.4 UPS Database Overview 1-6
1.4.1 UPS Database Files 1-6
1.4.2 UPS Database Structure 1-7
1.5 Using UPS Without a Database 1-7
1.6 UPS and UPD Commands 1-8
1.6.1 Syntax 1-8
1.6.2 Defaults 1-8
1.7 The UPS Environment 1-9
1.7.1 Initializing the UPS Environment 1-9
1.7.2 Changes UPS Makes to your Environment 1-10

Chapter 2: UPS Operations for the End User 2-1

2.1 Determining your Machine's Flavor 2-1
2.2 Listing Product Information in a Database 2-2
2.2.1 Formatted Output Style 2-3
2.2.2 Condensed Output Style 2-3
2.2.3 Examples 2-4
2.3 Finding a Product's Dependencies 2-7

2.4 Setting up a Product 2-8
2.4.1 The setup Command for the Typical Case 2-9
2.4.2 When You Need to Specify Other Options 2-9
2.5 Running Unsetup on a Product 2-10

Part II: Product Installer's Guide

(Part II is listed is listed in the front section of the table of contents.)

Chapter 3: General Product Installation Information 3-1
Chapter 4: Finding Information about Products on a Distribution Node . 4-1
Chapter 5: Installing Products Using UPD 5-1
Chapter 6: Installing Products Using UPP 6-1
Chapter 7: Installing Products using FTP 7-1
Chapter 8: Product Installation: Special Cases 8-1
Chapter 9: Troubleshooting UPS Product Installations 9-1

Part III: System Administrator's Guide

(Part III is listed is listed in the front section of the table of contents.)

Chapter 10: Maintaining a UPS Database 10-1
Chapter 11: UPS and UPD Pre-install Issues and General Administration 11-1
Chapter 12: Providing Access to AFS Products 12-1
Chapter 13: Bootstrapping CoreFUE 13-1
Chapter 14: Automatic UPS Product Startup and Shutdown 14-1

Part IV: Product Developer's Guide

Chapter 15: UPS Product Development: General Considerations 15-1
15.1 Product Development Considerations and Recommendations 15-1
15.1.1 All Products (Locally Developed and Third Party) 15-1
15.1.2 Products that You Develop 15-2
15.1.3 Third-Party Products Requiring a Hard-Coded Path 15-3

15.2 Tools for Developing and/or Packaging Products 15-5
15.2.1 Buildmanager 15-5
15.2.2 CVS 15-5
15.2.3 Template_product 15-6
15.3 Directory Structure for a UPS Product Instance 15-6
Chapter 16: Building UPS Products 16-1
16.1 Basic Steps for Making a UPS Product 16-1
16.1.1 Build the Directory Hierarchy 16-2
16.1.2 Create the Table File 16-2
16.1.3 Declare the Product to your Development UPS Database 16-2
16.1.4 Copy the Product Executable to the bin Directory 16-3
16.1.5 Provide Product man Pages 16-3
16.1.6 Test the Product 16-4
16.2 Specifics for Different Categories of Products 16-4
16.2.1 Unflavored Scripts 16-4
16.2.2 Pre-built Binaries 16-5
16.2.3 Products Requiring Build (In-House and Third-Party) 16-6
16.2.4 Overlaid Products 16-7
16.3 Sample Auxiliary Files 16-8
16.3.1 README 16-8
16.3.2 INSTALL_NOTE 16-9
16.3.3 RELEASE_NOTES 16-9
Chapter 17: Making Products Available For Distribution 17-1
17.1 Product Distribution Overview 17-1
17.2 Creating Product Tar Files 17-2
17.3 Adding a Product 17-3
17.3.1 Product Categories Defined for KITS 17-3
17.3.2 Examples 17-4
17.4 Adding an Independent Table File 17-5
17.5 Replacing a Component (Table File or ups Directory) 17-6
17.6 Adding/Changing a Chain 17-7
17.7 Deleting a Product or Component 17-8
17.8 Cloning a Product 17-8
17.9 Including Source in one of Fermilab's CVS Repositories 17-9
17.10 Product Announcement Policies 17-10
Chapter 18: Using template_product to Build and Distribute UPS Products 18-1
18.1 Overview 18-1
18.2 Accessing template_product 18-2
18.3 Cloning template_product 18-2
18.4 The Top-Level Makefile 18-3
18.5 Inserting your Product into the Template 18-4

18.6 Building the Product 18-4
18.6.1 Add Build Instructions 18-4
18.6.2 Run the Initial Build 18-4
18.6.3 Add Build Instructions to Top-Level Makefile 18-4
18.6.4 Rebuild Instructions 18-5
18.7 Testing your Product 18-5
18.8 Customizing your Tar File 18-5
18.9 Adding your Product to a Distribution Node 18-6
18.9.1 Add Product to fnkits 18-7
18.9.2 Specify Multiple Flavors 18-7
18.10 Adding your Product Source to a CVS Repository 18-8
18.11 Removing your Product from a Distribution Node 18-8

Chapter 19: Checklist for Building and Distributing Products 19-1

19.1 Pre-build Checklist 19-1
19.2 Build the Product 19-2
19.3 Test the Product 19-2
19.4 Distribute to fnkits as “test” 19-3
19.5 Announce the Product 19-3
19.6 Distribute to fnkits as “current” 19-4

Part V: Distribution Node Maintainer’s Guide

(Part V is listed is listed in the front section of the table of contents.)

Chapter 20: Product Distribution Server Configuration 20-1

Chapter 21: Configuration of the fnkits Product Distribution Node 21-1

Part VI: UPS and UPD Command Reference

Chapter 22: UPS Command Reference 22-1

22.1 setup 22-3
22.1.1 Command Syntax 22-3
22.1.2 Commonly Used Options 22-3
22.1.3 All Valid Options 22-3
22.1.4 More Detailed Description 22-5
22.1.5 setup Examples 22-6
22.2 unsetup 22-9
22.2.1 Command Syntax 22-9
22.2.2 All Valid Options 22-9
22.2.3 More Detailed Description 22-11
22.2.4 unsetup Examples 22-12

22.3 ups configure 22-13
22.3.1 Command Syntax 22-13
22.3.2 Commonly Used Options 22-13
22.3.3 All Valid Options 22-13
22.3.4 More Detailed Description 22-15
22.3.5 ups configure Examples 22-15
22.4 ups copy 22-17
22.4.1 Command Syntax 22-17
22.4.2 Commonly Used Options 22-17
22.4.3 All Valid Options 22-17
22.4.4 Options Valid with -G 22-19
22.4.5 More Detailed Description 22-19
22.4.6 ups copy Examples 22-20
22.5 ups declare 22-21
22.5.1 Command Syntax 22-21
22.5.2 Commonly Used Options 22-21
22.5.3 All Valid Options 22-22
22.5.4 More Detailed Description 22-24
22.5.5 ups declare Examples 22-26
22.6 ups depend 22-27
22.6.1 Command Syntax 22-27
22.6.2 Commonly Used Options 22-27
22.6.3 All Valid Options 22-27
22.6.4 ups depend Examples 22-29
22.7 ups exist 22-31
22.7.1 Command Syntax 22-31
22.7.2 Commonly Used Options 22-31
22.7.3 All Valid Options 22-31
22.7.4 More Detailed Description 22-33
22.7.5 ups exist Examples 22-33
22.8 ups flavor 22-35
22.8.1 Command Syntax 22-35
22.8.2 Commonly Used Options 22-35
22.8.3 All Valid Options 22-35
22.8.4 More Detailed Description 22-36
22.8.5 ups flavor Examples 22-37
22.9 ups get 22-39
22.9.1 Command Syntax 22-39
22.9.2 All valid options 22-39
22.9.3 ups get Example 22-40
22.10 ups help 22-41
22.10.1 ups help Example 22-41
22.11 ups list 22-43
22.11.1 Command Syntax 22-43
22.11.2 Commonly Used Options 22-43
22.11.3 All Valid Options 22-43

22.11.4 More Detailed Description 22-45
22.11.5 ups list Examples 22-49
22.12 ups modify 22-55
22.12.1 Command Syntax 22-55
22.12.2 Commonly Used Options 22-55
22.12.3 All Valid Options 22-55
22.12.4 More Detailed Description 22-56
22.12.5 ups modify Example 22-57
22.13 ups start 22-59
22.13.1 Command Syntax 22-59
22.13.2 Commonly Used Options 22-59
22.13.3 All Valid Options 22-59
22.13.4 More Detailed Description 22-61
22.13.5 ups start Examples 22-61
22.14 ups stop 22-63
22.14.1 Command Syntax 22-63
22.14.2 Commonly Used Options 22-63
22.14.3 All Valid Options 22-63
22.14.4 More Detailed Description 22-65
22.14.5 ups stop Examples 22-65
22.15 ups tailor 22-67
22.15.1 Command Syntax 22-67
22.15.2 Commonly Used Options 22-67
22.15.3 All Valid Options 22-67
22.15.4 More Detailed Description 22-69
22.15.5 ups tailor Example 22-69
22.16 ups touch 22-71
22.16.1 Command Syntax 22-71
22.16.2 Commonly Used Options 22-71
22.16.3 All Valid Options 22-71
22.16.4 ups touch Example 22-72
22.17 ups unconfigure 22-73
22.17.1 Command Syntax 22-73
22.17.2 Commonly Used Options 22-73
22.17.3 All Valid Options 22-73
22.17.4 More Detailed Description 22-75
22.17.5 ups unconfigure Example 22-75
22.18 ups undeclare 22-77
22.18.1 Command Syntax 22-77
22.18.2 Commonly Used Options 22-77
22.18.3 All Valid Options 22-78
22.18.4 More Detailed Description 22-79
22.18.5 ups undeclare Examples 22-80
22.19 ups verify 22-81
22.19.1 Command Syntax 22-81
22.19.2 Commonly Used Options 22-81

22.19.3 All Valid Options 22-81
22.19.4 ups verify Example 22-83
Chapter 23: UPD/UPP Command Reference 23-1
23.1 upd addproduct 23-3
23.1.1 Command Syntax 23-3
23.1.2 Commonly Used Options 23-4
23.1.3 All Valid Options 23-4
23.1.4 More Detailed Description 23-7
23.1.5 Adding Products to fnkits.fnal.gov 23-8
23.1.6 upd addproduct Examples 23-9
23.2 upd cloneproduct 23-11
23.2.1 Command Syntax 23-11
23.2.2 All Valid Options 23-11
23.2.3 Options Valid with -G 23-12
23.2.4 upd cloneproduct Example 23-12
23.3 upd delproduct 23-13
23.3.1 Command Syntax 23-13
23.3.2 Commonly Used Options 23-13
23.3.3 All Valid Options 23-13
23.3.4 upd delproduct Example 23-14
23.4 upd depend 23-15
23.4.1 Command Syntax 23-15
23.4.2 Options 23-15
23.4.3 upd depend Examples 23-15
23.5 upd exist 23-17
23.5.1 Command Syntax 23-17
23.5.2 Options 23-17
23.5.3 upd exist Examples 23-17
23.6 upd fetch 23-19
23.6.1 Command Syntax 23-19
23.6.2 Commonly Used Options 23-19
23.6.3 All Valid Options 23-19
23.6.4 upd fetch Examples 23-21
23.7 upd get 23-23
23.7.1 Command Syntax 23-23
23.7.2 Options 23-23
23.8 upd install 23-25
23.8.1 Command Syntax 23-25
23.8.2 Commonly Used Options 23-25
23.8.3 All Valid Options 23-25
23.8.4 Options Valid with -G 23-28
23.8.5 More Detailed Description 23-28
23.8.6 upd install Examples 23-29

23.9 upd list 23-31
23.9.1 Command Syntax 23-31
23.9.2 Options 23-31
23.9.3 upd list Examples 23-31
23.10 upd modproduct 23-33
23.10.1 Command Syntax 23-33
23.10.2 Commonly Used Options 23-33
23.10.3 All Valid Options 23-34
23.10.4 More Detailed Description 23-35
23.10.5 upd modproduct Examples 23-36
23.11 upd repproduct 23-39
23.11.1 Command Syntax 23-39
23.11.2 Options 23-40
23.11.3 upd repproduct Examples 23-40
23.12 upd update 23-41
23.12.1 Command Syntax 23-41
23.12.2 Commonly Used Options 23-41
23.12.3 All Valid Options 23-41
23.12.4 upd update Examples 23-43
23.13 upd verify 23-45
23.13.1 Command Syntax 23-45
23.13.2 Options 23-45
23.14 upp 23-47
23.14.1 Command Syntax 23-47
23.14.2 All Valid Options 23-47
23.14.3 upp Examples 23-47

Chapter 24: Generic Command Option Descriptions 24-1

24.1 Alphabetical Option Listing 24-1
24.2 More Information on Selected Options 24-7
24.2.1 -e 24-7
24.2.2 -H 24-7
24.2.3 -K 24-7
24.2.4 -q 24-8
24.2.5 -V 24-9

Chapter 25: UPS/UPD Command Usage 25-1

25.1 Syntax 25-1
25.1.1 Order of Command Line Elements 25-1
25.1.2 Specifying Version/Chain 25-1
25.1.3 Grouping Option Flags 25-2
25.1.4 Specifying Arguments to Options 25-2
25.1.5 Embedded Spaces in Option Arguments 25-2
25.1.6 Invalid Option Arguments 25-3
25.1.7 Specifying Multiple Products in a Single Command 25-3

25.1.8 Multiple Occurrences of Same Option Flag 25-3
25.1.9 Use of Wildcards 25-4
25.2 Options 25-4
Chapter 26: Product Instance Matching in UPS/UPD Commands 26-1
26.1 Database Selection Algorithm 26-1
26.1.1 UPS 26-1
26.1.2 UPD 26-2
26.2 Instance Matching within Selected Database 26-3
26.2.1 Where Does Instance Matching Take Place? 26-3
26.2.2 Flavor Selection 26-3
26.2.3 Qualifiers: Use in Instance Matching 26-4
26.2.4 Flavor and Qualifier Matching Algorithm 26-4

Part VII: Administrator's Reference

(Part VII is listed is listed in the front section of the table of contents.)

Chapter 27: Information Storage Format in Database and Configuration Files
Chapter 28: Version Files 28-1
Chapter 29: Chain Files 29-1
Chapter 30: The UPS Configuration File 30-1
Chapter 31: The UPD Configuration File 31-1
Chapter 32: The UPP Subscription File 32-1

Part VIII: Developer's Reference

Chapter 33: Actions and ACTION Keyword Values 33-1
33.1 Overview of Actions 33-1
33.2 UPS Command Actions 33-1
33.2.1 UPS Commands as Keyword Values 33-1
33.2.2 "Uncommands" as Keyword Values 33-2
33.3 Chain Actions 33-3
33.3.1 Chains as Keyword Values 33-3
33.3.2 "Unchains" as Keyword Values 33-3
33.4 The "Unknown Command" Handler 33-3
33.5 Actions Called by Other Actions 33-4
Chapter 34: Functions used in Actions 34-1
34.1 Overview of Functions 34-1
34.2 Reversible Functions 34-1
34.3 Function Descriptions 34-2
34.3.1 addAlias 34-2

34.3.2 doDefaults 34-3
34.3.3 envAppend 34-3
34.3.4 envPrepend 34-4
34.3.5 envRemove 34-4
34.3.6 envSet 34-5
34.3.7 envSetIfNotSet 34-5
34.3.8 envUnset 34-5
34.3.9 exeAccess 34-6
34.3.10 exeActionOptional 34-6
34.3.11 exeActionRequired 34-6
34.3.12 execute 34-7
34.3.13 fileTest 34-7
34.3.14 pathAppend 34-8
34.3.15 pathPrepend 34-8
34.3.16 pathRemove 34-9
34.3.17 pathSet 34-9
34.3.18 prodDir 34-9
34.3.19 setupEnv 34-10
34.3.20 setupOptional 34-10
34.3.21 setupRequired 34-10
34.3.22 sourceCompileOpt 34-11
34.3.23 sourceCompileReq 34-11
34.3.24 sourceOptCheck 34-12
34.3.25 sourceOptional 34-13
34.3.26 sourceReqCheck 34-13
34.3.27 sourceRequired 34-14
34.3.28 unAlias 34-14
34.3.29 unProdDir 34-14
34.3.30 unsetupEnv 34-15
34.3.31 unsetupOptional 34-15
34.3.32 unsetupRequired 34-16
34.3.33 writeCompileScript 34-16
34.4 Functions under Consideration for Future Implementation 34-17
34.5 Examples of Functions within Actions 34-18
34.5.1 A setup Action 34-18
34.5.2 A “declare as current” Action 34-18
34.6 Local Read-Only Variables Available to Functions 34-18
34.6.1 List of Current Read-Only Variables 34-19
34.6.2 Read-Only Variables under Consideration for the Future 34-21

Chapter 35: Table Files 35-1

35.1 About Table Files 35-1
35.2 When Do You Need to Provide a Table File? 35-1
35.3 Recommendations for Creating Table Files 35-2

35.4 Table File Structure and Contents 35-2
35.4.1 Basic Structure 35-2
35.4.2 Grouping Information 35-3
35.4.3 The Order of Elements 35-3
35.5 Product Dependencies 35-4
35.5.1 Defining Dependencies 35-4
35.5.2 Product Dependency Conflicts 35-4
35.6 Table File Examples 35-6
35.6.1 Example Illustrating Use of FLAVOR=ANY 35-6
35.6.2 Example Showing Grouping 35-6
35.6.3 Example with User-Defined Keywords 35-7
35.6.4 Examples Illustrating ExeActionOpt Function 35-8

Chapter 36: Scripts You May Need to Provide with a Product 36-1

36.1 configure and unconfigure 36-1
36.2 tailor 36-3
36.3 current and uncurrent 36-3
36.4 start and stop 36-3

Chapter 37: Use of Compile Scripts in Table Files 37-1

37.1 Overview 37-1
37.2 Usage Information 37-1

Chapter 38: Creating and Formatting Man Pages 38-1

38.1 Creating the Source Document (Unformatted) 38-2
38.1.1 Source File Format 38-2
38.1.2 Man Page Information Categories 38-3
38.1.3 Example Source File 38-4
38.2 Formatting the Source File 38-5
38.2.1 nroff 38-5
38.2.2 groff 38-6
38.3 Converting your Man Page to html Format 38-6

Glossary GLO-1

Index IDX-1

About this Manual

This chapter provides an introduction to the *Complete Guide and Reference Manual for UPS, UPD and UPP v4*. In particular you will find:

- the overall structure, the purpose and the intended audience of the manual
- what parts of the manual you need
- where to obtain this manual and where to look for updates
- the typeface conventions and symbols used throughout the document
- an invitation to readers to send us comments

This manual is published in three submanuals: GU0014A, GU0014B, and GU0014C. The structure of the document and its division into these sections is discussed in the following sections.

1. Document Structure, Purpose and Intended Audiences

The *UPS and UPD v4 Reference Manual* is intended for several different user groups as listed on the next page. To best accommodate the different types of users, the manual is divided into five user guides (Parts I-V):

- Part I *Overview and End User's Guide*
- Part II *Product Installer's Guide*
- Part III *System Administrator's Guide*
- Part IV *Product Developer's Guide*
- Part V *Distribution Node Maintainer's Guide*

and three reference manuals (Parts VI-VIII)

- Part VI *UPS and UPD Command Reference*
- Part VII *Administrator's Reference*
- Part VIII *Developer's Reference*

The user guides explain and illustrate the **UPS/UPD/UPP** tasks associated with each user group. The reference guides provide detailed information on commands, concepts, file structure/contents, and so on. On the following page is a guide to which parts of the manual you are likely to need, according to your job functions. Notice that we recommend Parts I and VI for all users:

Parts	User Functions
A: For All Users	
Part I *Overview and End User's Guide* Part VI *UPS and UPD Command Reference*	*End Users:* List product information in a **UPS** database on a user system; Access installed software products Access FermiTools[a] software products (*Other user groups' functions described later in table*)
B: For Product Installers, **UPS** Database Administrators, System Administrators of User Machines, Distribution Node Maintainers	
Part II *Product Installer's Guide*	*Product Installers:* Install software products from a **UPS** product distribution node into a **UPS** database on a user system; Install products into the AFS-space **UPS** database
Part III *System Administrator's Guide* and Part VII *Administrator's Reference*	*System Administrators, UPS Database Administrators:* Maintain **UPS** products in a **UPS** database; Install **UPS**/**UPD**/**UPP** on a user system; Configure **UPS** on a user system; Configure **UPD** on a user system; Configure **UPP** on a user system; Configure an installed product to start/stop automatically at boottime/shutdown
Part V *Distribution Node Maintainer's Guide*	*Distribution Node Maintainers:* Install **UPS**/**UPD** on a distribution system; Configure **UPS** and **UPD** on a distribution system; Configure Web and anonymous **FTP** servers on a distribution system Maintain **UPS** database on a distribution system
C: Product Developers	
Part IV *Product Developer's Guide* Part VIII *Developer's Reference*	*Product Developers and Maintainers:* Develop and maintain software products that are intended to be distributed in accordance with **UPS** standards; Adapt pre-existing or third-party software to conform to **UPS** standards; Distribute products

a. Fermilab-written software products that are made publicly available.

The table above lists rather generally the topics that the manual covers. Note that it is *not* the purpose of this document to provide information on:

- general UNIX system administration
- general UNIX or Fermilab information (see instead *UNIX at Fermilab*, GU0001)
- the use of any particular software product other than **UPS**/**UPD**/**UPP**

CDF and D0 collaborators: Also see *A UNIX Based Software Management System* (GU0013) at `http://www-cdf.fnal.gov/offline/code_management/run2_cmgt/run2_cmgt.html` to find information describing how **UPS** and **UPD** have been implemented in your experiments' code management systems.

2. Availability

Copies of the *UPS and UPD v4 Reference Manual* (GU0014A, B, and C), can be obtained from the following sources:

Web	`http://www.fnal.gov/docs/products/ups/ReferenceManual/` This can be accessed under **Documentation** on the Computing Division home page. Search using any of the following keywords: afs, develop(ment), distribute(tion), fermitools, GU0014, install(ation), kits, maintain(tenance), man page, product, system administration, unix, upd, upp, ups
Paper Copies	Wilson Hall, 8th floor, NE (just across from what used to be the Computing Division library)

3. Updates

Pending subsequent releases of this manual, updates will be maintained on the Web with the on-line version of the manual. To get there from the Computing Division home page, select **Documentation**, request *GU0014* and follow the pointers (see "Web" under section 2. *Availability*).

4. Conventions

The following notational conventions are used in this document:

bold	Used for product names (e.g., **UPS**).
italic	Used to emphasize a word or concept in the text. Also used to indicate logon ids and node names.
`typewriter`	Used for filenames, pathnames, contents of files, output of commands.

typewriter-bold	Used to indicate commands and prompts.
[...]	In commands, square brackets indicate optional command arguments and options.
|	When shown in a command example (e.g., **x|y|z**), separates a series of options from which one may or must be chosen (depends if enclosed in square brackets). In UNIX commands, used to pipe output of preceding command to the following one.
’ ... **’**	Single vertical quotes indicate apostrophes in commands.
" ... "	Double vertical quotes indicate double quotes in commands
...	In a command, means that a repetition of the preceding parameter or argument is allowed.
%	Prompt for C shell family commands (**%** is also used throughout this document when a command works for both shell families).
$	Prompt for Bourne shell family commands; also standard UNIX prefix for environment variables (e.g., $VAR means “the value to which VAR is set”).
****	UNIX standard quoting character; used in commands throughout the manual to indicate that the command continues to the next line
<...>	In commands, variables, pathnames and filenames, angle brackets indicate strings for which reader must make a context-appropriate substitution. For example, $<PRODUCT>_DIR becomes $EMACS_DIR for the product **emacs**.
{ }	In local read-only variables, e.g., ${UPS_PROD_DIR}, string should be used as shown with the {}.

All command examples are followed by an implicit carriage return key.

Some of the files discussed in this document are shell family-specific, and thus come in pairs. Their filenames carry the extensions `.sh` and `.csh`. We often refer to a pair of these files as `filename.[c]sh`.

The following symbols are used throughout this document to draw your attention to specific items in the text:

A “bomb”; this refers to something important you need to know in order to avoid a pitfall.

This symbol is intended to draw your attention to a useful hint.

5. Your Comments are Welcome!

The *UPS and UPD v4 Reference Manual* may contain some errors, however we endeavor to minimize the error count! We encourage all the readers of this document to report back to us:

- errors or inconsistencies that we have overlooked
- any parts of the manual that are confusing or unhelpful -- please offer *constructive* suggestions!
- other topics to include (keeping in mind the purpose of the manual)
- tricks, hints or ideas that other users might find helpful

Send your comments via email to *cdlibrary@fnal.gov.*

Part II Product Installer's Guide

Chapter 3: *General Product Installation Information*

This chapter provides general information you need to know before you start installing products. It discusses:

- the three ways to install **UPS** products (**UPD**, **UPP** and **FTP**)
- how to register your node to download products from `KITS`
- how to determine where **UPD** installs products on your system
- how to declare a product instance to a database manually
- some questions that can come up during an installation
- post-installation procedures required for some products
- how to handle networking restrictions at off-site locations

Chapter 4: *Finding Information about Products on a Distribution Node*

This chapter discusses finding information about products on a distribution node, in particular:

- how to find out which **UPS** products are available on a distribution node
- how to list a product's dependencies as declared on the distribution node
- product file permissions and pathnames for downloading products from *fnkits* via **FTP**
- special instructions for downloading proprietary products from *fnkits*

Chapter 5: *Installing Products Using UPD*

This chapter guides you through installing products from a **UPS/UPD** product distribution node using the **UPD** command **`upd install`**.

Chapter 6: *Installing Products Using UPP*

UPP can be used for several functions as described briefly in section 1.1 *Introduction to UPS, UPD and UPP*, and in detail in Chapter 32: *The UPP Subscription File*. This chapter describes how to use **UPP** to install products.

Chapter 7: *Installing Products using FTP*

This chapter describes how to download a product using **FTP**, install it, and declare it to a local **UPS** database.

Chapter 8: *Product Installation: Special Cases*

This chapter provides product installation information about specific cases. It discusses:

- how to install products requiring special privileges
- how to install into a local products area using the installation of **UPD** in AFS space
- how to install products into the AFS-space **UPS** products area

Chapter 9: *Troubleshooting UPS Product Installations*

This chapter provides a few hints if things don't seem to work after installing a product.

Chapter 3: General Product Installation Information

This chapter provides general information you need to know before you start installing products. It discusses:

- the three ways to install **UPS** products (**UPD**, **UPP** and **FTP**)
- how to register your node to download products from `KITS`
- how to determine where **UPD** installs products on your system
- how to declare a product instance to a database manually
- some questions that can come up during an installation
- post-installation procedures required for some products
- how to handle networking restrictions at off-site locations

Installing products into AFS space is not covered in this chapter; see section 8.3 *Installing Products into AFS Space*.

3.1 Installation Methods for UPS Products

There are three ways to access products from a **UPS** product distribution node: using **UPD**, **FTP** or **UPP** (which is actually a layer on top of **UPD**). Each method is described briefly below, and then in more detail in the following chapters. Information on troubleshooting a problematic product installation is provided in Chapter 9: *Troubleshooting UPS Product Installations*.

3.1.1 UPD

The **UPD** product includes the **`upd install`** command for installing products. This is the most widely-used product installation method on machines running **UPS/UPD**. Chapter 5: *Installing Products Using UPD* is dedicated to describing this process. Installation parameters are set in the local node's **UPD** configuration. The aspects of the configuration that you as a product installer need to be aware of are described in section 3.3 *What You Need to Know about Your System's UPD Configuration*; the **UPD** configuration is described in detail in Chapter 31: *The UPD Configuration File*.

The **`upd install`** command performs the following functions:

- retrieves the specified product instance, and by default its dependencies, from a distribution node
- unwinds the product (if transferred in tar format) and installs it, and by default its dependencies, on the user node according to the node's **UPD** configuration

- declares the product, and by default its dependencies, to the database specified in the node's **UPD** configuration
- either resolves dependencies or prints to screen the commands you will need to issue in order to do so

3.1.2 UPP

UPP is a layer on top of **UPD** that can be used to perform a variety of tasks, as described in Chapter 32: *The UPP Subscription File*. Regarding product installation, it can be configured to run `upd install` for specified products under specified conditions. Dependencies of the specified products are updated automatically, as well, so that the integrity of the products is maintained. **UPP** can also be given instructions to run the necessary `ups declare` commands to resolve dependencies when a product installation finishes. **UPP** can be run manually, or it can be automated using a tool like **cron**. Chapter 6: *Installing Products Using UPP* illustrates how to use it to install products.

3.1.3 FTP

Anonymous **FTP** is available on *fnkits*, and may be available on other **UPS** product distribution nodes. **FTP** does not take advantage of the **UPD** configuration. It can be used only to retrieve products; it is left to the installer to unwind and declare them. Furthermore, if the table file and/or the `ups` directory is (are) not included the tar file, it (they) must be retrieved separately. Chapter 7: *Installing Products using FTP* is describes using **FTP** to install products.

FTP is not recommended for installations into the usual product area; **UPD** is designed and configured specifically for that and should be used instead. **FTP** is more suited to product installations into non-standard locations on your node, e.g., into your own area for use just by you.

On *fnkits*, **FTP** is most useful for off-site users who want to download FermiTools products, which are located under the `/pub` directory. You do not need to be a registered user to obtain the FermiTools products.[1]

3.2 User Node Registration for KITS

In order to download most products from the `KITS` database, the machine you're using must be registered with *fnkits.fnal.gov*. All machines in the `fnal.gov` domain are automatically registered. Off-site machines need to register using the **Product Distribution Platform Registration Request** form at
`http://www.fnal.gov/cd/forms/upd_registration.html`.

1. All machines in the `fnal.gov` domain are automatically registered to download products from `KITS`. Off-site machines need to register using the **Product Distribution Platform Registration Request** form at
`http://www.fnal.gov/cd/forms/upd_registration.html`.

If you only want to download FermiTools products, which are located under the `/pub` directory in `KITS`, you do not need to be using a registered node. FermiTools are made available to the general public.

3.3 What You Need to Know about Your System's UPD Configuration

When you install a product using **UPD** (or **UPP**), the installation parameters are controlled by the **UPD** configuration. The **UPS** configuration file for the database you're using points to a **UPD** configuration file. These configuration files described in Chapter 30: *The UPS Configuration File* and Chapter 31: *The UPD Configuration File*. The **UPD** configuration file typically consists of one or more stanzas, each of which:

- identifies certain product instances, products or groups of products
- specifies a database on the local system in which to declare a product matching the identifier
- specifies locations on the local system in which **UPD** is to put a matched product and its related files
- (optionally) lists actions for **UPS/UPD** to perform either just before or just after declaring the product

3.3.1 Location of UPD Configuration File

The Default UPD Configuration File

The **UPD** configuration file is stored as:

`${UPD_USERCODE_DIR}/updconfig`

where the keyword UPD_USERCODE_DIR is set in the **UPS** configuration file. It tells you the location of the database containing the **UPD** configuration file. When **UPD** gets setup, the read-only variable ${UPD_USERCODE_DIR} gets defined and set to the same value as the keyword. (The read-only variable ${UPD_USERCODE_DB} also gets defined and set to the database directory containing ${UPD_USERCODE_DIR}). To find the value of UPD_USERCODE_DIR, you can list the **UPS** configuration file, e.g.,:

```
% less $PRODUCTS/.upsfiles/dbconfig
```

or you can first setup **UPD**, and the request the variable value, e.g.,:

```
% echo $UPD_USERCODE_DIR
```

or

```
% env | grep UPD
```

Overriding the Default UPD Configuration

If your system is set up with multiple **UPS** databases configured to point to different **UPD** configurations, you can choose to specify a database on the **`upd install`** command line pointing to a **UPD** configuration file other than the default. First, verify that the database you specify points to the **UPD** configuration you want. To find out, run the command:

```
% ups list -z <database> -K UPD_USERCODE_DIR
```

Note that if this command returns empty quotes, it means the database specifies *no* configuration file. In this case the default **UPD** configuration will not be overridden.

3.3.2 Where Products Get Declared

The keyword UPS_THIS_DB, set in the **UPD** configuration file, identifies the database into which **UPS** declares the product (i.e., the directory that **UPD** specifies in the **`ups declare -z <database>`** option). This keyword may be set differently in different stanzas, thereby causing different products to be declared in different databases.

3.3.3 Where Products Get Installed

For organizational reasons it is usually preferable to have **UPD** configured to install all the **UPS** products for a database in one area. In the **UPS** configuration file, typically the keyword PROD_DIR_PREFIX gets set to the product root directory prefix under which the products reside. The **UPD** configuration file then defines product root directory locations in terms of PROD_DIR_PREFIX. The quantities you need to be aware of within the **UPD** configuration file are:

UPS_PROD_DIR	The product root directory. The **`upd install`** command runs the **`ups declare`** command and uses this value as the argument to the **`-r`** option. It is usually defined relative to PROD_DIR_PREFIX.
UNWIND_PROD_DIR	The absolute path to directory where products get unwound. In most cases, it's the product root directory (in terms of read-only variables: `${PROD_DIR_PREFIX}/${UPS_PROD_DIR}`), however in AFS and some NFS mounting configurations, products are often unwound and installed in different locations (see section 8.3 *Installing Products into AFS Space*).

You should not specify the product location in the **`upd install`** command unless you want to override the default.

3.4 Declaring an Instance Manually

A product instance must exist on the system before it can be declared to a **UPS** database[1]. Product declaration is done with the **`ups declare`** command. Declaring a product instance makes it known to **UPS**, and therefore retrievable within the **UPS** framework. Normally products are installed on user nodes using the **`upd install`** command which, in addition to downloading and installing the product, runs **`ups declare`** to make the initial declaration of the product to the local **UPS** database. If you use **FTP** to download a product, then you'll need to declare it manually. Refer to Chapter 7: *Installing Products using FTP* for details about installing with **FTP**.

If you use **`upd install`** and you have more than one database, refer to section 5.2 *How UPD Selects the Database* to see how **UPD** determines the database for the declaration.

3.4.1 The ups declare Command

Before declaring, make sure the product is unwound into in its final location. Also make sure that you've downloaded the table file and installed it in an appropriate directory. For an initial declaration you must specify at a minimum: the product name, product version, product root directory, flavor and table file name[2].

The full command description and option list is in the reference section 22.5 *ups declare*. Here we show commonly used command options (see the notes regarding **`-z`**, **`-U`** and **`-M`** which follow):

```
% ups declare <product> <version> -r /path/to/prod/root/dir/ \
  -f <flavor> [-z /path/to/database] [-U /path/to/ups/dir] \
  [-m <table_name>.table] [-M /path/to/table/file/dir] \
  [<chainFlag>]
```

1) If the database is not specified using **`-z`**, **UPS** declares the product into the first listed database in $PRODUCTS (see section 26.1 *Database Selection Algorithm* for more information).

2) If the product's `ups` directory tar file was unwound in the default location (`$<PRODUCT>_DIR/ups`), then **`-U /path/to/ups/dir`** is not needed. If the `ups` directory is located elsewhere (or named differently), this specification must be included. If specified as a *relative* path, it is taken as relative to the product root directory.

1. At least a rudimentary root directory hierarchy for the product, its table file directory and table file must exist before declaration.
2. Two exceptions: (1) if the product consists only of a table file that sets up a list of dependencies, there is no product root directory; and (2) if the product has no table file (very rare) then there is no table file name.

3) If the product's table file was placed in either of the two default locations (under `/path/to/database/<product>/` or in the product's `ups` directory), then **`-M /path/to/table/file/dir`** is not needed. Only use the **`-M`** option if you have moved the table file to a separate location where **UPS** won't otherwise find it. If specified as a *relative* path, it is taken as relative to the product root directory. See section 28.4 *Determination of ups Directory and Table File Locations* for details on how **UPS** finds the table file.

Unless the product you're declaring has no table file (true for very few products), make sure its location gets declared properly, either explicitly or by default. Otherwise, users will need to specify its name and location on the command line every time they want to run or operate on the product. If it is neither declared nor specified on the command line, **UPS/UPD** assumes there is no table file.

You can opt to declare a chain to the product instance at this time or in a later declaration. To declare a chain, include the appropriate chain flag in the command (see section 1.3.5 *Chains* for a listing).

3.4.2 Examples

For more examples see the reference section 22.5 *ups declare.*

Declaration of New Product to Non-default Database

The following command shows a fairly typical product declaration. We'll install a product called **histo** v4_0 onto a SunOS+5 node. We assume the product instance's `ups` directory is maintained under its product root directory, and that it contains the table file. We include the **`-z`** option to indicate that we want to override the default database selection. This is the first instance of this product to be declared to this database, therefore the **`ups declare`** command automatically creates the appropriate product directory under the specified database:

```
% ups declare histo v4_0 -f SunOS+5 -m histo.table -z $MY_DB -r\
  /path/to/products/SunOS+5/histo/v4_0
```

We can run a **`ups list -l`** command to see all the declaration information (include **`-a`** because it's not yet declared current):

```
% ups list -alz $MY_DB histo
```

```
DATABASE=/path/to/ups_database/declared
        Product=histo   Version=v4_0    Flavor=SunOS+5
                Qualifiers=""   Chain=""
                Declared="1998-04-17 22.08.30 GMT"
                Declarer="aheavey"
                Modified="1998-04-17 22.08.30 GMT"
                Modifier="aheavey"
                Home=/path/to/products/SunOS+5/histo/v4_0
                No Compile Directive
                Authorized, Nodes=*
                UPS_Dir="ups"
                Table_Dir=""
                Table_File="v4_0.table"
                Archive_File=""
                Description=""
                Action=setup
                        prodDir()
                        setupEnv()
                        addalias(histo,${UPS_PROD_DIR}/bin/histo)
```

```
addalias(hsdir,${UPS_PROD_DIR}/bin/hsdir)
envSet(HISTO_INC,${UPS_PROD_DIR}/include)
```

Declaration of Additional Instance of a Product

In the following example we declare an additional instance of **histo**, of the same version, but for the flavor IRIX+5. Again the table file resides under the product root directory's `ups` subdirectory, and we override the default database. This time we declare it with the chain "test" (**`-t`**):

```
% ups declare histo v4_0 -tf IRIX+5 -m histo.table -z $MY_DB -r\
  /path/to/products/IRIX+5/histo/v4_0
```

Running a **`ups list -a`** to see what the database now contains for this product, we find:

```
% ups list -az $MY_DB histo
```

```
DATABASE=/path/to/ups_database/declared
        Product=histo   Version=v4_0    Flavor=SunOS+5
                Qualifiers=""   Chain=""

        Product=histo   Version=v4_0    Flavor=IRIX+5
                Qualifiers=""   Chain=test
```

Declaration with Table File Located in Database

Depending on your configuration, you may want the table file to reside in the product's subdirectory under the database (e.g., `$PRODUCTS/<product>/<table_file>`).

A table file for the product must be placed in this location before the instance is declared to the database. Therefore, if you are declaring the first instance of a product to the database, you need to manually create the product directory under the database and copy the table file into it before declaring the instance.

You still do not need to specify the table file location (**`-M`** option) on the **`ups declare`** command line; **UPS** will find it here.

3.5 Installation FAQ

3.5.1 What File Permissions Get Set?

Product files get downloaded and installed with the same permissions that they have on the distribution node, minus the **umask** set in your login files. We recommend that you set your **umask** to **`002`** before installing any products to ensure that you don't remove the group write access for table files.

3.5.2 You're Ready to Install: Should you Declare Qualifiers?

If a product instance is declared with one or more qualifiers on the distribution node, you can choose whether you want to declare it on your system with or without them. If you don't need the qualifiers in order to distinguish between different copies of the same product, it's usually easiest to declare products without them. Otherwise, users must enter the qualifiers on the command line exactly as they appear in the product declaration each time they want to setup that product instance or perform other **UPS** operations on it. The files that **UPS** uses to manage each product allow comment lines (see section 27.1 in Chapter 27: *Information Storage Format in Database and Configuration Files*); this provides a way of recording qualifier information if you choose not to declare the qualifiers explicitly.

3.5.3 What if an Install Gets Interrupted?

Normally **UPD** deletes the installed portions of a product when an installation process gets interrupted, and it doesn't declare the pieces that failed to install. Therefore, you generally don't need to worry about cleaning up before reattempting the installation. Just issue the install command again, the same way as you did the first time.

However, if you interrupted the process for some reason (e.g., you saw it was running out of space), then you'll need to remove by hand the piece that was being installed at the time of the interruption. How will you know? Reattempt the install, and if you get a message similar to this:

```
directory /a/b/c already exists, will not overwrite.
```

then you'll need to remove the specified directory/file(s).

3.5.4 What if a Product was Installed under a Different Name?

Giving a product a new name upon installation can cause problems in dependency trees. This practice is not supported, and is certainly not encouraged, but it can be made to work. If you have a product that needs to find, for example, $MYPROD_DIR, but **myprod** has been installed on your system with a different name, e.g., **fermi_myprod**, then you may need to edit the table file (described in Chapter 35: *Table Files*). Normally product installers never need to touch the table file, but this is an exception. If the provided table file for **myprod** was written by a developer who has no knowledge of the name change on your system, the table file probably contains:

```
ACTION=SETUP
   prodDir()
```

where `prodDir()` instructs the **`setup`** command to set the variable $<PRODUCT>_DIR (see 22.1.4 *More Detailed Description* under section 22.1 *setup*). On your system then, the variable $FERMI_MYPROD_DIR will get set, but $MYPROD_DIR won't. To ensure that you also get the variable $MYPROD_DIR, edit the table file and under ACTION=SETUP add the function:

```
envSet(MYPROD_DIR, ${UPS_PROD_DIR})
```

3.6 Post-Installation Procedures

Some products require that you perform supplementary steps during or after the installation process, for example copying files to other locations or creating needed files or directories. The product's `INSTALL_NOTE` file should contain any instructions for completing the installation. Commonly required actions on the installer's part include configuring and/or tailoring the product instance.

3.6.1 Configuring a Product

Post-installation procedures that can be completely automated are typically collected together such that the command **`ups configure`** executes them. This command gets executed by default, as necessary, when the product instance is declared. Otherwise, you can run **`ups configure`** manually at any time after declaration to configure the product instance.

The configuration may involve creating links to the product root directory from other areas (see section 8.1 *Installing Products that Require Special Privileges*). If the area is not identical for each node (i.e., same path but separate areas) accessing the **UPS** database in which the product instance has been declared, then you will need to run the **`ups configure`** command manually on each node that mounts a unique area. If you are not sure whether you need to configure a product instance on each node, look through the configuration steps in the table file under ACTION=CONFIGURE to see what they do.

3.6.2 Tailoring a Product

Tailoring is the aspect of the product implementation that requires input from the product installer (e.g., specifying the location of hardware devices for a software driver package). If the product requires tailoring, a file is usually supplied in the format of an interactive executable (script or compiled binary), and it is run via the **UPS** command **`ups tailor`**. You must explicitly tailor the product instance using **`ups tailor`**; tailoring is *not* performed automatically.

Tailoring is generally allowed on any node of a cluster, however we strongly recommend that you perform any node-specific tailoring from that node, or flavor-specific tailoring from a node of that flavor to avoid mismatches.

3.7 Networking Restrictions at your Site

Some off-site locations may impose networking restrictions which can interfere with **UPD**.

3.7.1 Proxying Webserver

If all web traffic is channeled through a proxying webserver at your site, you need to provide the URL of this server to **UPD**. Since **UPD** commands go through a web server, they will fail otherwise (the error message will indicate either "Destination unreachable" or "Timeout").

Look at your web browser configuration to find out what proxy you're using. In **Netscape** this would be under **EDIT/PREFERENCES.../ADVANCED/PROXIES**. Set the variable http_proxy (lower case) to the URL of your server, e.g., (for C shell):

```
% setenv http_proxy "http://some.host.name:8000"
```

3.7.2 Firewall for Incoming TCP Connections

If your site firewalls incoming TCP connections, but allows outgoing ones, you'll need to set the FTP_PASSIVE variable to the value 1, e.g., (for C shell):

```
% setenv FTP_PASSIVE 1
```

This will make **UPD** use "passive mode" **FTP** transfers.

Chapter 4: Finding Information about Products on a Distribution Node

This chapter discusses finding information about products on a distribution node, in particular:

- how to find out which **UPS** products are available on a distribution node
- how to list a product's dependencies as declared on the distribution node
- product file permissions and pathnames for downloading products from *fnkits*[1] via **FTP**
- special instructions for downloading proprietary products from *fnkits*

4.1 Listing Products on a Distribution Node

Both **UPD** and **UPP** can be used to list the product instances installed in a distribution database. We show you how to do this below.

For information on the `KITS` database (on *fnkits*) or the products in AFS space, you can also use the Web. Fill out and submit the form at `http://www.fnal.gov/upc/`. This page can be accessed from the Computing Division's documentation search page (`http://cddocs.fnal.gov/cfdocs/productsDB/docs.html`; scroll down and click on *OSS Product Status Request Page* under **Additional Information**). See the newsgroups `fnal.announce.products` and `fnal.announce.unix` for recent product release information.

4.1.1 Using UPD

The **`upd list`** command is available to list information about product instances on the server. It just performs **`ups list`** (described in section 2.2 *Listing Product Information in a Database*) on a distribution database, and uses the same set of options. Two output styles are provided: a formatted one that is easy for users to read, and a condensed one for parsing by a subsequent command or a script.

You can specify the information you want contained in the output by including various options (see the reference section 22.11 *ups list* for details). As is standard in **UPS/UPD**, if no chain, version or flavor is specified, and **`-a`** (for all instances) is not specified, it returns only the instance declared as current for the best-matched flavor of the requesting machine. All the **UPD** commands use the *fnkits* host as the default, which accesses the `KITS` database. Use the **`-h <host>`** option to specify a different host.

1. *fnkits.fnal.gov* is the Computing Division's central product server. Its distribution database is generally referred to as `KITS`.

The **upd list** command has the following syntax:

```
% upd list [<options>] [<product>] [<version>]
```

Most of the sample commands listed below use the **-K+** option which condenses the standard output onto a single line. The **-K+** option is described in section 2.2.2 *Condensed Output Style*.

List Current Instance of a Product

Most often, people want to know if there is a current version of a product that will run on their machine. Because of the defaults in place, you can issue the simple command:

```
% upd list [-K+] [<product>]
```

For example, take the product **tex**. To request the condensed output, enter:

```
% upd list tex -K+
    "tex" "v3_14159" "SunOS+5" "" "current"
    "tex" "v3_1415a" "SunOS" "" "current"
```

List All Instances for One Flavor of a Product

Say, for example, you wanted to know what instances of **ximagetools** were available from *fnkits* for the flavor SunOS+5. You could issue the command:

```
% upd list -af SunOS+5 ximagetools -K+
```

```
    "ximagetools" "v3_0b" "SunOS+5" "" ""
    "ximagetools" "v3_1" "SunOS+5" "" "old"
    "ximagetools" "v3_1_1" "SunOS+5" "" ""
```

List All Current Products for Flavor of Machine

For this request, you may just want the product name and version. Use the **-K** option accordingly (output edited for brevity):

```
% upd list -Kproduct:version
```

```
    "acnet" "v1_0"
    "alerts" "v0_1"
    "apache" "v1_3_3"
    "bash" "v2_02"
    ...
    "ximagetools" "v4_0"
    "xntp" "v3_5_93"
    "xntp" "v3_4"
    "xpdf" "v0_5"
```

Obtain Detailed Listing for a Product Instance

To find all the information associated with a product instance on the server, use the **`-l`** option (output edited for brevity):

```
% upd list -l tex v3_14159
```

```
DATABASE=/ftp/upsdb
        Product=tex     Version=v3_14159        Flavor=SunOS+5
                Qualifiers=""   Chain=current
                Declared="1998-09-10 07.39.47 GMT:1998-09-10 07.39.47 GMT"
                Declarer="updadmin:updadmin:updadmin:updadmin"
                Modified="1998-09-10 07.39.47 GMT:1998-09-10 07.39.47 GMT"
                Modifier="updadmin:updadmin:updadmin:updadmin"
                Home=/ftp/products/tex/v3_14159/SunOS+5/tex_v3_14159_SunOS+5
                ...
                Action=setup
                        setupRequired(tex_files)
                        prodDir()
                        setupEnv()
                        pathPrepend(PATH,${UPS_PROD_DIR}/bin)
                        envSet(TEXMFMAIN,${UPS_PROD_DIR}/share/texmf)
```

4.1.2 Using UPP

UPP requires what we call a *subscription file* which tells it what products to look for on a designated distribution node, and what functions to perform when it detects that new versions of these products have been released there. One of the functions **UPP** can perform is notification. It is therefore a useful tool for keeping abreast of changes/enhancements to your favorite products and to get information on new ones.

Your job is to create a **UPP** subscription file and run the **`upp`** command. The subscription file has a structure that includes a header and at least one "stanza". A stanza is bracketed by **`begin`** and **`end`**. Each product you want to monitor requires its own stanza (or a separate subscription file, but that is more cumbersome). The **`upp`** command can be automated and run periodically (for example from **cron**). The information below illustrates how to write a subscription file for the current purpose. For more information, refer to Chapter 6: *Installing Products Using UPP* and Chapter 32: *The UPP Subscription File*.

Sample Subscription File

This example includes a header (the first five lines in the sample file) as required, and one stanza. This file configures **UPP** to send email to a specified address when it detects either of two conditions on the designated distribution server:

- any new product has been added
- a version of **exmh** of flavor SunOS+5.5 has been chained to "current"

The file listing is on the left and comments are provided on the right:

`file = upp`	This identifies the file as a **UPP** subscription file.
`mail_address = joe@fnal.gov`	Send mail notifications to *joe@fnal.gov*.
`dist_node = fnkits.fnal.gov`	Use `fnkits.fnal.gov` as the **UPD** server node to query.

`data_dir = /var/adm/upp`	Use `/var/adm/upp` as the local bookkeeping directory
`newprod_notify = T`	`T` is for True; this means yes, notify me of new products appearing on the **UPD** server node (in this case, on the *fnkits* node). Set it to any other value (e.g., `F`) to disable this notification.
`begin`	Begin stanza for a product.
`product = exmh`	Identify subscribed product as **exmh** (the **exmh** versions remain unspecified in this example, therefore act on all versions for the flavor specified below). You must include this `product` identifier in the stanza. Optional product instance identifiers not used in this example include `qualifiers`, `prod_dir` (product root directory) and `chain`.
`flavor = SunOS+5.5`	Identify flavor of product (this is optional)
`action = current`	List in the following lines of this file one or more functions to perform when any version of **exmh** of flavor SunOS+5.5 is chained to "current" on *fnkits*. `Action` can be set to any chain name, or to the value `newversion`. `Newversion` means: perform the following functions when a new version of the product appears on the server.
`notify`	Send a notification message to *joe@fnal.gov*
`end`	End stanza. If you want to add instructions for another product in this same file, start a new stanza with `begin`.

The UPP Command

You can run **UPP** interactively by issuing the **`upp`** command. The **`upp`** command line is very simple:

`% upp [-v[v...]] <subscript-file-1> [<subscript-file-2>...]`

The **`-v`** option requests verbose output; more **`v`**'s (up to four) provide progressively more verbosity. The **`upp`** command has no direct output (unless verbosity is turned on), rather it mails a report of any actions taken to the email address specified in the subscription file.

There are no other command options for **`upp`**; its behavior is controlled entirely by the subscription file(s).

Automate UPP Using cron

You can add a **cron** job that first sets up **UPD** then runs **UPP** with a subscription file (shown here as `upp.subscription`). Here is a sample **sh** script to which we give the filename `upp.launch`:

```
#! /bin/sh
. /usr/local/etc/setups.sh
setup upd
upp /path/to/upp.subscription
```

A sample **crontab** entry to run the `upp.launch` script every night at midnight might look like:

```
0 0 * * * /path/to/upp.launch
```

4.2 Listing Product Dependencies on a Distribution Node

The command **`upd depend`** is used to return the list of dependencies for the specified product instance(s), as declared in the database in the distribution database. It just performs **`ups depend`** (described in section 2.3 *Finding a Product's Dependencies*) on the server, and uses the same set of options. As usual, the *fnkits* node is the default, and **`-h <host>`** allows you to specify a different server. This example shows several layers of dependencies:

`% upd depend exmh`

```
exmh v2_0_2 -f NULL -z /ftp/upsdb -g current
|__expect v5_25 -f SunOS+5 -z /ftp/upsdb -g current
|  |__tk v8_0_2 -f SunOS+5 -z /ftp/upsdb
|     |__tcl v8_0_2 -f SunOS+5 -z /ftp/upsdb
|__mh v6_8_3c -f SunOS+5 -z /ftp/upsdb -g current
|  |__mailtools v2_3 -f NULL -z /ftp/upsdb -g current
|__mimetools v2_7a -f SunOS+5 -z /ftp/upsdb -g current
|__glimpse v3_0a -f SunOS+5 -z /ftp/upsdb -g current
|__www v3_0 -f NULL -z /ftp/upsdb -g current
|  |__lynx v2_8_1 -f SunOS+5 -z /ftp/upsdb -g current
|__ispell v3_1b -f SunOS+5 -z /ftp/upsdb -g current
```

Inclusion of the **`-R`** option returns only the required dependencies; any optional ones are ignored:

`% upd depend -R exmh`

```
exmh v2_0_2 -f NULL -z /ftp/upsdb -g current
|__expect v5_25 -f SunOS+5 -z /ftp/upsdb -g current
|  |__tk v8_0_2 -f SunOS+5 -z /ftp/upsdb
|     |__tcl v8_0_2 -f SunOS+5 -z /ftp/upsdb
|__mh v6_8_3c -f SunOS+5 -z /ftp/upsdb -g current
|  |__mailtools v2_3 -f NULL -z /ftp/upsdb -g current
|__mimetools v2_7a -f SunOS+5 -z /ftp/upsdb -g current
```

Another useful option to point out is **`-H <flavor>`** which allows you to retrieve a dependency list for a flavor other than that of the machine you're using. For example, if you want the dependency list for an IRIX+6 version, but you issue the command from a SunOS+5 machine, you would run the command:

`% upd depend -H IRIX+6 exmh`

```
exmh v2_0_2 -f IRIX+6 -z /ftp/upsdb -g current
|__expect v5_25 -f IRIX+5 -z /ftp/upsdb -j  -g current
|  |__tk v8_0_2 -f IRIX+5 -z /ftp/upsdb -j  -g current
|__tk v8_0_2 -f IRIX+5 -z /ftp/upsdb -j  -g current
|  |__tcl v8_0_2 -f IRIX+5 -z /ftp/upsdb -j  -g current
|__tcl v8_0_2 -f IRIX+5 -z /ftp/upsdb -j  -g current
|__mh v6_8_3c -f IRIX+6 -z /ftp/upsdb -g current
|  |__mailtools v2_3 -f NULL -z /ftp/upsdb -g current
|__mimetools v2_7a -f IRIX+6 -z /ftp/upsdb -g current
|__glimpse v3_0a -f IRIX+6 -z /ftp/upsdb -g current
```

```
|__www v3_0 -f NULL -z /ftp/upsdb -g current
|  |__lynx v2_8_1 -f IRIX+6 -z /ftp/upsdb -g current
|__ispell v3_1b -f IRIX+6 -z /ftp/upsdb -g current
```

4.3 Information about Products in KITS

KITS is the commonly used name for the distribution database on the Computing Division's central product distribution server, *fnkits.fnal.gov.* In this section we describe:

- registering your node to download products
- the permissions set on products in this database
- the directory hierarchy of the products area

4.3.1 Access Restrictions and Product Categories

The permissions/access restrictions of products in KITS depend on the category of product. As a product installer, you don't generally know (or need to know) a priori what category the product belongs to, but if you can't download a particular product due to access restrictions, the product category is probably the reason.[1] The categories as defined in the **UPD** configuration for KITS are:

default	The default category is the most commonly used, and is for regular products that are intended for distribution to **FTP** clients on registered hosts[2]. The products are set to group upd, and group-read-only.
fermitools	fermitools products are locally-developed and supported software packages (which are not available elsewhere, generally) that are made available to the public via the FermiTools program[3]. These products are world-readable, and thus accessible by any **FTP** client.
proprietary	The proprietary category includes products for which Fermilab has a limited number of licenses. Each proprietary product has its own group, and is made group-readable only to that group.[4] To gain access, appropriate "site group" and "site gpass" commands must be issued by the **FTP** client.

1. You can download the **optionlist** table file to find a product's category. See section 20.6.5 *Flagging Special Category Products Using Optionlist*.
2. See the **Product Distribution Platform Registration Request** form at http://www.fnal.gov/cd/forms/upd_registration.html.
3. For more information on FermiTools, see http://www.fnal.gov/fermi-tools/.
4. In general, the group name for the proprietary product is the same as the product name, except that all **vxworks**-related products share a group.

`fnalonly`	The `fnalonly` category is for products intended only for on-site **ftp** clients in the `fnal.gov` domain coming in through `ftp://ftp.fnal.gov:9021/`. These products are set to group `fnalonly`, and are group-read-only.
`usonly`	US-only (United States only) products are accessible only to U.S. government (`.gov`) and military (`.mil`) domains coming in through `ftp://ftp.fnal.gov:8021/`. In general, these are products for which distribution to other countries is illegal. They are set to group `usonly`, and are group-read-only.

4.3.2 Product Pathnames for FTP Access

Products are arranged (via symlinks) in several different file hierarchies to make browsing easier:

- The `/ftp/products` directory contains products organized by product name and version.
- The `/ftp/KITS` hierarchy contains products organized by operating system. This is the "old-style" hierarchy.
- The `/ftp/pub` hierarchy contains the FermiTools products which are available to the general public.

/ftp/products Area

Within **FTP**, `cd` to `ftp/products` (or to just `products`) in order to access this hierarchy. Under the `/ftp/products` area the product tar files are organized in the structure:

```
<product>/<version>/<flavor>/
```

Product tar files are named according to this convention (intentionally missing the underscore between flavor and qualifiers):

```
<product>_<version>_<flavor><qualifiers>.tar
```

Shown on two lines for better readability, the whole path is:

```
/ftp/products/<product>/<version>/<flavor>/
     <product>_<version>_<flavor><qualifiers>.tar
```

For example, the tar file for the product **xemacs** version v20_4 for SunOS+5 is maintained under `/ftp/products` at:

```
/ftp/products/xemacs/v20_4/SunOS+5/xemacs_v20_4_SunOS+5.tar
```

The product **vxworks** provides an example with a qualifier (`68k`) tacked onto the end:

```
/ftp/products/vxworks/v5_3c/SunOS+5/vxworks_v5_3c_SunOS+568k.tar
```

/ftp/KITS Area

Within **FTP**, `cd` to `ftp/KITS` (or to just `KITS`) in order to access this hierarchy. Under the `/ftp/KTIS` area the product directories are organized in the structure:

```
<base_flavor>/<product>/<version>/<flavor>/
```

Here you will find links to the product tar files under the `/ftp/products` structure.

/ftp/KITS/pub Area

Within **FTP**, **`cd`** to `ftp/KITS/pub` (or to `KITS/pub` or just `pub`) in order to access this hierarchy. Under the `/ftp/KTIS` area the product directories are organized in the structure:

```
<product>/<version>/
```

There is no directory level for flavor. Here you will find links to product tar files under the `/ftp/products` structure.

4.4 Special Instructions for Proprietary Products

Some products in `KITS` are flagged as "proprietary" and require a group id and password for installation. This allows us to distribute products in a more controlled fashion. You don't need to know ahead of time if a given product falls into this category; when you attempt to install a proprietary product, the system will return a message of the form (shown for the product **edt**):

```
Product edt v6_3b SunOS+5 is a proprietary product.
Before it can be installed, you need to obtain a group name and password for
it by sending a proprietary products request form to compdiv@fnal.gov

Have you obtained a group name and password? n
Do you need a proprietary product request form? y
informational: transferred proprietary.form
        from fnkits.fnal.gov:/ftp/products/proprietarylist to
        /your/home/directory/proprietary.form
```

If you request the form, as shown, you will find it downloaded to your home directory in ASCII format for easy editing. Fill out the form and email it to the address listed in the form. Another option is to fill out and submit the Web-based form at:

`http://www.fnal.gov/cd/forms/proprietary_form.html`

If your request is approved, you will receive email with a valid group id and password for the product. You can then install the product, entering these two items when prompted (it's best to cut and paste the group id and password from the email onto your terminal window to prevent typos[1]), e.g.,:

```
Product edt v6_3b SunOS+5 is a proprietary product.
...
Have you obtained a group name and password? y

Enter Group Name: edt
Enter Group Pass word: somepassword
informational: beginning install of edt.
```

1. One case in which it is particularly useful to cut and paste these items is the package **vxworks**, which has multiple proprietary parts. Fortunately the same group id and password is used for each part, but you do need to enter it several times.

The installation proceeds as normal from this point on. If you enter the group id and password incorrectly, or if they have expired (which happens within a few days after they are sent), you will get an error message like the following:

```
...
Enter Group Name: wrong
Enter Group Pass word: wrong
informational: beginning install of edt.
ftp SITE GPASS failed: Group access request incorrect.

error: can't transfer /ftp/products/edt/v6_3b/SunOS+5/edt_v6_3b_SunOS+5
         from fnkits.fnal.gov to
        /fnal/ups/prd/edt/v6_3b/SunOS+5
```

If this message appears, try again to be sure the values get entered correctly. If it really doesn't work, reply to the email you received containing the group id and password, and ask to have the product re-opened for you.

Chapter 5: Installing Products Using UPD

This chapter guides you through installing products from a **UPS/UPD** product distribution node using the **UPD** command **`upd install`**. **UPD** is the most efficient and widely-used product installation method on machines running **UPS/UPD**. The installation parameters are set in the local node's **UPD** configuration. The aspects of the **UPD** configuration that you as a product installer need to be aware of are described in section 3.3 *What You Need to Know about Your System's UPD Configuration*; the configuration file itself is described in detail in Chapter 31: *The UPD Configuration File*.

5.1 The upd install Command

The **`upd install`** command performs the following functions:

- retrieves the specified product instance, and by default its dependencies, from a distribution node
- installs the product, and by default its dependencies, on the user node according to the local **UPD** configuration
- unwinds the product(s) if transferred in tar format
- declares the product, and by default its dependencies, to the database specified in the local **UPD** configuration. You may pass options to this local database declaration.
- either resolves dependencies (if the **`-X`** option is specified) or prints to screen the commands you will need to issue in order to do so

5.1.1 Command Syntax

The full description of the **`upd install`** command and the options it takes can be found in the reference section 23.8 *upd install*. The command syntax, showing some commonly used options, is:

```
% upd install [<chainFlag>] [-G "<options>"]  [-h <host>] \
  [-H <flavor>] [-q <qualifiers>] [-X] [-z <databaseList>] \
  [<other options>] <product> [version]
```

5.1.2 Passing Options to the Local ups declare Command

The **-G** option allows you to pass **UPS** options to the local **ups declare** command, which gets called internally by **upd install**. It accepts multiple options. The elements valid for use with **-G** include **<product>**, **<version>** and the following subset of the **ups declare** options:

-A <nodeList>, **-c**, **-d**, **-D <origin>**, **-f <flavor>**, **-g <chainName>**, **-n**, **-o**, **-O "<flagList>"**, **-p "<description>"**, **-q <qualifierList>**, **-t**, **-z <databaseList>**, **-0**, **-1**, **-2**, **-3**

This feature is most commonly used to declare a chain to the product, e.g., the "current" chain:

```
% upd install -G "-c" [<other options>] <product> [<version>]
```

The **-G** construction can also be used to reset identifying information like flavor and qualifiers. For example, to download the OSF1+V3 version of a product with qualifier **oldxyz**, but declare it locally as OSF1 with qualifier **newxyz**, use the **-f** (or **-H**), **-q** and **-G** options as shown:

```
% upd install -H OSF1+V3 -q oldxyz -G "-f OSF1 -q newxyz" \
  [<other options>] <product> [<version>]
```

5.2 How UPD Selects the Database

5.2.1 Database Selection Algorithm

upd install runs **ups declare** on the local node to declare the product instance to a local database. The local **UPD** configuration is always used to determine the database into which products must be installed/declared. If your **UPS** installation includes only one database, that is the one into which all declarations will go (assuming that the **UPD** configuration used by that database points to it, which is generally the case).

If there are multiple databases, **upd install** has to determine the database for the declaration. In a nutshell, it:

1) picks a starting database

   To pick the starting database, **UPD** first looks to see if the **-z <databaseList>** option is specified on the **upd install** command line. If so, **UPD** picks the first database listed there. If not, **UPD** picks the first database listed in $PRODUCTS.

2) finds the **UPD** configuration file to which the database points[1]

3) looks in this **UPD** configuration to see where to install and declare the product

If your local **UPS/UPD** installation is particularly complicated, it might be useful to verify that your $PRODUCTS variable includes all the databases used by the **UPD** configuration(s), and only those. If not, it is possible to declare products into a database not listed in $PRODUCTS, which generally is undesirable.

---

1. The location of the `updconfig` file is set via the keyword UPD_USERCODE_DIR in the database's `dbconfig` file.

5.2.2 Database Selection for Dependencies

This works the same way as it does for the main product. For each dependency in turn, **UPD** looks in the **UPD** configuration file designated by the first database it encounters. From the **UPD** configuration, it determines the database in which to declare the dependent product, and where to install the product files. (If the database already contains a declaration for the same instance of the product, the product does not get reinstalled/redeclared.)

5.2.3 Selecting a Database for Development or Testing

For development and/or testing purposes, it is often convenient to install products in your own products area and declare them in your own database, separate from the working database(s) on your system. Setting up your own database is discussed in section 11.8.2 *Adding a New Database and/or Products Area*.

If you're working in AFS space or in an NIS cluster with its own common NFS-mounted database, also see TN0091 *Configuring a Local UPS Database (While Still Using the Centrally Supported AFS database)* at `http://www.fnal.gov/docs/TN/tn0091.html`., or section 12.2 *Configuring a Local Database to Work With AFS*.

Even if you've prepended your database path to $PRODUCTS, when you're ready to install a product and declare it in your database, remember that you may need to use the **`-z <database>`** option in the **`upd install`** command, as discussed above.

5.3 Checklist for Installing a Product using UPD

The procedural list below is a full checklist for a product installation, including pre- and post-install checks. The checks are not strictly necessary, the list simply provides guidelines for monitoring a product installation. Having said that, it's always a good idea to at least get a "snapshot" of your database before and after each product installation to aid in troubleshooting in the event the installation is not entirely successful. Include options/arguments as necessary in the suggested commands.

1) Run **`upd list`** to verify that the desired product instance is available on the server.

2) Run **`upd depend`** to list the product's dependencies (lists both required and optional by default).

3) Run **`upd depend -R`** to list only its required dependencies (compare to the **`upd depend`** output to determine the optional ones).

4) Run **`ups list`** to see which if any of the dependencies already exist in the local target database.

5) Run **`upd install`** to install the product instance and, as desired, its required and optional dependencies.

When testing/troubleshooting, you might want to use the **`-i`** option to ignore errors, or the **`-v`**(**`vvv`**) option to produce verbose output.

6) Run **`ups declare -c`** if you want the parent product to be “current” but you did not include **`-G "-c"`** in **`upd install`**.

7) Run commands to resolve dependencies, if indicated in the output of the **`upd install`** command.

8) Run **`ups list`** for the parent product and dependencies to verify the declarations.

9) Setup the parent product and test that it works.

5.4 Examples

5.4.1 Install a Product Using Default Database

This illustrates the simplest case: installing the current instance of a product for the best-match flavor of the target machine, and letting **UPS** determine the target database using $PRODUCTS. For this example, we choose a product with no dependencies. First, check the default instance on the server:

```
% upd list -K+ teledata
```

```
"teledata" "v1_0" "NULL" "" "current"
```

Check which instances already exist in the database(s) listed in $PRODUCTS:

```
% ups list -aK+ teledata
```

(*if no output, then no instances*)

Install the default instance. We are not passing any arguments to the local **`ups declare`** command (no **`-G`** option).

```
% upd install teledata
```

```
informational: installed teledata v1_0.
informational: product teledata has an INSTALL_NOTE;
        you should read /export/home/t1/aheavey/upsII/products/teledata/v1_0//up
s/INSTALL_NOTE.
```

Read the `INSTALL_NOTE` file to see if you need to do anything (we’ll not document this part since it is product-specific). Next, verify that the instance is now declared in $PRODUCTS:

```
% ups list -aK+ teledata
```

```
"teledata" "v1_0" "NULL" "" ""
```

Redeclare the instance with the current chain:

```
% ups declare -c teledata v1_0
```

Verify that the instance is now declared as current:

```
% ups list -aK+ teledata
```

```
"teledata" "v1_0" "NULL" "" "current"
```

5.4.2 Install a Product, Specifying Database

Perform the installation normally, (as shown in section 5.4.1 *Install a Product Using Default Database*) but include the **`-z`** option, e.g.,:

```
% upd install -z $MYDB teledata [<other options>]
```

or

```
% upd install -z /path/to/my/database teledata [<other options>]
```

assuming $MYDB is set to `/path/to/my/database`. The main product and all of its dependencies, if any, are installed and declared according to the **UPD** configuration to which the specified database points.

5.4.3 Install a Product and All Dependencies

By default, **`upd install`** installs the specified (parent) product and all of its dependencies. It checks for the presence of the dependencies as discussed in section 5.2 *How UPD Selects the Database*, and installs each as necessary, skipping over the ones already there. Make sure that you include neither the **`-j`** nor the **`-R`** option on the **`upd install`** command line. The use of these two options is illustrated in the following sections, 5.4.4 *Install a Product and No Dependencies* and 5.4.5 *Install a Product and Required Dependencies Only*.

To illustrate how **UPD** handles dependencies that are already installed versus those that aren't, we'll first install two of **pine**'s six dependencies separately, and then install **pine** v4_05 itself. We start with an empty database. First list **pine**'s dependencies:

```
% upd depend pine
```

```
pine v4_05 -f IRIX+6 -z /ftp/upsdb -g current
|__ispell v3_1b -f IRIX+6 -z /ftp/upsdb -g current
|__ximagetools v4_0 -f NULL -z /ftp/upsdb -g current
   |__imagelibs v1_0 -f IRIX+6 -z /ftp/upsdb
   |__imagemagick v4_04 -f IRIX+6 -z /ftp/upsdb
   |__xfig v3_20 -f IRIX+6 -z /ftp/upsdb<---- already there
   |__xanim v2_70_64 -f IRIX+6 -z /ftp/upsdb
```

We'll install **xfig** and **xanim** ahead of time, without specifying a chain:

```
% upd install xfig -H IRIX+6
```

```
% upd install xanim -H IRIX+6
```

Install **pine** and chain it to current:

```
% upd install pine -H IRIX+6 -G "-c"
```

```
informational: xanim v2_70_64 already exists on local node, skipping.
informational: xfig v3_20 already exists on local node, skipping.
informational: installed imagemagick v4_04.
informational: installed imagelibs v1_0.
informational: installed ximagetools v4_0.
informational: installed ispell v3_1b.
informational: installed pine v4_05.
```

Take a snapshot of the post-installation database:

```
% ups list -aK+
```

```
"imagelibs" "v1_0" "IRIX+6" "" "current"
"imagemagick" "v4_04" "IRIX+6" "" "current"
"ispell" "v3_1b" "IRIX+6" "" "current"
"pine" "v4_05" "IRIX+6" "" "current"
"xanim" "v2_70_64" "IRIX+6" "" ""
"xfig" "v3_20" "IRIX+6" "" ""
"ximagetools" "v4_0" "NULL" "" "current"
```

Notice that all the products are chained to current except the two that were preinstalled with no chain. The installer should go ahead and declare them current, too, so that the main product recognizes them as dependencies[1].

A second example illustrates how **UPS/UPD** resolves chains and discusses the effect of the **-X**, **-s** and **-v** options. We install **www** version v2_7b. First list its dependencies:

```
% upd depend www v2_7b
```

```
www v2_7b -f SunOS+5 -z /ftp/upsdb
|__ximagetools v4_0 -f NULL -z /ftp/upsdb -g current
|  |__imagelibs v1_0 -f SunOS+5 -z /ftp/upsdb
|  |__imagemagick v4_04 -f SunOS+5 -z /ftp/upsdb
|  |__xfig v3_20 -f SunOS+5 -z /ftp/upsdb
|  |__xanim v2_70_64 -f SunOS+5 -z /ftp/upsdb
|__xpdf v0_5 -f SunOS+5 -z /ftp/upsdb -g current
```

Let's assume none of these products exists in our local database, and run the install:

```
% upd install www v2_7b
```

```
informational: installed xpdf v0_5.
informational: installed xanim v2_70_64.
informational: installed xfig v3_20.
informational: installed imagemagick v4_04.
informational: installed imagelibs v1_0.
informational: installed ximagetools v4_0.
informational: installed www v2_7b.
informational: product www has an INSTALL_NOTE;    you should read
 /local/ups/prd/upd/mengel/test/out_products/www/SunOS-5/v2_7b/ups/INSTALL_NOTE.
Execute the following to resolve chains:
ups declare -f SunOS+5 -q "" -g current xpdf v0_5
        -z /local/ups/prd/upd/mengel/test/out_ups_database
ups declare -f NULL -q "" -g current ximagetools v4_0
        -z /local/ups/prd/upd/mengel/test/out_ups_database
```

If we had included the **-X** option, **UPS/UPD** would have executed these two **ups declare** commands. If we had run it with the **-s** option, which just lists what the command would do, the output would have looked like:

```
informational: would have installed xpdf v0_5.
informational: would have installed xanim v2_70_64.
informational: would have installed xfig v3_20.
informational: would have installed imagemagick v4_04.
informational: would have installed imagelibs v1_0.
informational: would have installed ximagetools v4_0.
informational: would have installed www v2_7b.
upd install would have succeeded.
```

1. Typically dependencies are defined by chain rather than by version.

If we had run it with the **`-v`** (verbose) option and **`-X`**, we would have seen (output edited for brevity):

```
informational: beginning install of xpdf.
informational: transferred /ftp/products/xpdf/v0_5/SunOS+5/xpdf_v0_5_SunOS+5
        from fnkits.fnal.gov to
        /local/ups/prd/upd/mengel/test/out_products/xpdf/SunOS-5/v0_5
informational: transferred /ftp/products/xpdf/v0_5/SunOS+5/xpdf_v0_5_SunOS+5/ups
/.
        from fnkits.fnal.gov to
        /local/ups/prd/upd/mengel/test/out_products/xpdf/SunOS-5/v0_5/ups
informational: transferred /ftp/products/xpdf/v0_5/SunOS+5/xpdf_v0_5_SunOS+5.tab
le
        from fnkits.fnal.gov:/ to
        /local/ups/prd/upd/mengel/test/out_ups_database/xpdf/v0_5.table.new
informational: ups declare succeeded
. . . (plus similar output for each remaining product) . . .
```

5.4.4 Install a Product and No Dependencies

Perform the installation normally (as shown in section 5.4.1 *Install a Product Using Default Database*), but include the **`-j`** option, e.g.,:

```
% upd install -j <product> [<version>] [<other options>]
```

The specified product gets installed, but none of its dependencies do.

5.4.5 Install a Product and Required Dependencies Only

In this example we'll install the product **exmh** version v2_0_2, flavor IRIX+6 (the default for our system) and its required dependencies only. We perform the installation normally, but include the **`-R`** option to specify "required dependencies only". First take a snapshot of the local database into which the product and its required dependencies will be declared:

```
% ups list -aK+
```

```
"imagelibs" "v1_0" "IRIX+6" "" "current"
"imagemagick" "v4_04" "IRIX+6" "" "current"
"ispell" "v3_1b" "IRIX+6" "" "current"
"pine" "v4_05" "IRIX+6" "" "current"
"xanim" "v2_70_64" "IRIX+6" "" ""
"xfig" "v3_20" "IRIX+6" "" ""
"ximagetools" "v4_0" "NULL" "" "current"
```

Check the full dependency list for this product:

```
% upd depend exmh
```

```
exmh v2_0_2 -f IRIX+6 -z /ftp/upsdb -g current
|__expect v5_25 -f IRIX+5 -z /ftp/upsdb -j  -g current
|  |__tk v8_0_2 -f IRIX+5 -z /ftp/upsdb -j  -g current
|__tk v8_0_2 -f IRIX+5 -z /ftp/upsdb -j  -g current
|  |__tcl v8_0_2 -f IRIX+5 -z /ftp/upsdb -j  -g current
|__tcl v8_0_2 -f IRIX+5 -z /ftp/upsdb -j  -g current
|__mh v6_8_3c -f IRIX+6 -z /ftp/upsdb -g current
|  |__mailtools v2_3 -f NULL -z /ftp/upsdb -g current
```

```
|__mimetools v2_7a -f IRIX+6 -z /ftp/upsdb -g current
|__glimpse v3_0a -f IRIX+6 -z /ftp/upsdb -g current
|__www v3_0 -f NULL -z /ftp/upsdb -g current
|  |__lynx v2_8_1 -f IRIX+6 -z /ftp/upsdb -g current
|__ispell v3_1b -f IRIX+6 -z /ftp/upsdb -g current
```

Of these dependencies, only **ispell** v3_1b is already installed locally. List the required dependencies:

```
% upd depend -R exmh
```

```
exmh v2_0_2 -f IRIX+6 -z /ftp/upsdb -g current
|__expect v5_25 -f IRIX+5 -z /ftp/upsdb -j  -g current
|  |__tk v8_0_2 -f IRIX+5 -z /ftp/upsdb -j  -g current
|__tk v8_0_2 -f IRIX+5 -z /ftp/upsdb -j  -g current
|  |__tcl v8_0_2 -f IRIX+5 -z /ftp/upsdb -j  -g current
|__tcl v8_0_2 -f IRIX+5 -z /ftp/upsdb -j  -g current
|__mh v6_8_3c -f IRIX+6 -z /ftp/upsdb -g current
|  |__mailtools v2_3 -f NULL -z /ftp/upsdb -g current
|__mimetools v2_7a -f IRIX+6 -z /ftp/upsdb -g current
```

Glimpse, **www**, **lynx** and **ispell** are not in this list and therefore **`upd install -R`** should not install them. Install **exmh** (output not shown):

```
% upd install exmh v2_0_2 -H IRIX+6 -R
```

Now take a post-installation snapshot of the local database:

```
% ups list -aK+
```

```
"exmh" "v2_0_2" "IRIX+6" "" "current"
"expect" "v5_25" "IRIX+5" "" "current"
"imagelibs" "v1_0" "IRIX+6" "" "current"
"imagemagick" "v4_04" "IRIX+6" "" "current"
"ispell" "v3_1b" "IRIX+6" "" "current"
"mailtools" "v2_3" "NULL" "" "current"
"mh" "v6_8_3c" "IRIX+6" "" "current"
"mimetools" "v2_7a" "IRIX+6" "" "current"
"pine" "v4_05" "IRIX+6" "" "current"
"tcl" "v8_0_2" "IRIX+5" "" "current"
"tk" "v8_0_2" "IRIX+5" "" "current"
"xanim" "v2_70_64" "IRIX+6" "" ""
"xfig" "v3_20" "IRIX+6" "" ""
"ximagetools" "v4_0" "NULL" "" "current"
```

Notice that **exmh**, all of its required dependencies, and none of its optional ones are listed (except **ispell** which was already there).

Chapter 6: Installing Products Using UPP

UPP can be used for several functions as described briefly in section 1.1 *Introduction to UPS, UPD and UPP*, and in detail in Chapter 32: *The UPP Subscription File*. This chapter describes how to use **UPP** to install products.

6.1 Overview of Using UPP to Install Products

UPP requires what we call a *subscription file* which tells it what products to look for on a designated distribution node, and what functions to perform when it detects that new versions of these products are released there. One of the functions **UPP** can perform is product installation. **UPP** does this by running `upd install` (described in Chapter 5: *Installing Products Using UPD*). You can also instruct **UPP** to run `ups declare` commands to resolve dependencies as necessary when a product installation finishes.

Your job is to create a **UPP** subscription file and run the `upp` command. To automate **UPP**'s operations, the `upp` command can be run periodically (for example from **cron**) .

6.2 Creating a UPP Subscription File

A subscription file consists of a header followed by at least one stanza. The header includes an email address for notification, the distribution node to query, and other "administrative" information. Each stanza has three parts:

- identification of a product or particular instances of a product
- identification of the condition(s) for which you want **UPP** to perform the instructions you give it
- a list of instructions, or functions to perform, for each condition

A stanza is bracketed by the lines `begin` and `end`. The number of stanzas per file is not limited. A stanza cannot refer to multiple products, however there can be multiple stanzas for the same product (e.g., for treating different instances of the same product differently).

6.2.1 Create the Header

The header should look similar to this example (explanations are on the right):

`file = upp`	This identifies the file as a **UPP** subscription file.
`mail_address = joe@fnal.gov`	This specifies the email address to which **UPP** is to send notifications.
`dist_node = fnkits.fnal.gov`	This specifies the product distribution node to contact.
`data_dir = /var/adm/upp`	This refers to the directory where you want **UPP** to maintain bookkeeping files.
`newprod_notify = T`	Setting `newprod_notify` to `T` (True) tells **UPP** to send notification of brand new products to the address in `mail_address`.

6.2.2 Identify the Product

Within a stanza, the following terms can be used in matching a new or updated product instance: `product`, `flavor`, `version`, `qualifiers`, `prod_dir` (product root directory), and `chain`. Set them to values that you want **UPP** to monitor on the distribution node.

All instances that match a given set of values will be operated on (in contrast to the standard **UPS** and **UPD** matching algorithms; see Chapter 26: *Product Instance Matching in UPS/UPD Commands*). You can specify only the product name and thereby install all instances, or restrict the set of instances by specifying more information. Most of the time, you only need to specify product (and sometimes flavor). An example of this part of the file is:

```
begin
  product = exmh
  flavor = SunOS+5.5
  ...
end
```

6.2.3 Trigger the Product Installation

After identifying the product instance, you need to tell **UPP** when to install it on your system. Your choices are when a new version of the product appears on the distribution node, or when the product on the distribution node gets chained to a value that matches your specification. This gets done in an *action* line, e.g.,

```
action = newversion
```

or

```
action = current
```

Any chain, including user-defined chains, can be specified.

6.2.4 Provide Instructions to UPP

At this point you're ready to tell **UPP** what to do when the conditions are met. Since this chapter discusses installing products, the instructions you can choose from are:

`install`	Install the subscribed product via **`upd install`**.
`reget`	Short for: delete, then reinstall
`resolve`	Run any **`ups declare`** commands as necessary to make chains match so that parent product and dependencies and run properly together.
`notify`	Place a notice of the new product instance in the mailed output.

6.3 Sample Subscription File for Installing a Product

This sample file instructs **UPP** to install all the SunOS+5.5 instances of the product **exmh** (and dependencies as necessary), and to resolve the dependencies. **UPP** is also instructed to send notification when the install is triggered. The file contents are on the left, and explanations on the right:

`file = upp`	This identifies the file as a **UPP** subscription file.
`mail_address = joe@fnal.gov`	Send mail notifications to *joe@fnal.gov.*
`dist_node = fnkits.fnal.gov`	Use `fnkits.fnal.gov` as the **UPD** server node to contact
`data_dir = /var/adm/upp`	Use `/var/adm/upp` as the bookkeeping directory
`newprod_notify = T`	Yes (True), notify me of new products appearing on the **UPD** server node (in this case, on the *fnkits* node).
`begin`	Begin stanza for a product.
`product = exmh`	Identify subscribed product as **exmh** (the **exmh** versions remain unspecified in this example, therefore act on all versions for the flavor specified below).
`flavor = SunOS+5.5`	Identify flavor of product (this is optional)
`action = current`	List in the following lines one or more functions to perform when an instance of **exmh** of flavor SunOS+5.5 is chained to current on *fnkits*.
`notify`	Send a notification message to *joe@fnal.gov*

`install`	Install the newly current instance (and its dependencies as necessary) on the local node
`resolve`	Determine which `ups declare` commands need to be run on the local node so that all the chains match up properly for the dependencies to work, then run the commands.
`end`	End. If you want to add instructions for another product in this same file, start a new stanza with "begin".

6.4 The UPP Command

The **`upp`** command line is very simple:

`% upp [-v[v...]] <subscrip_file_1> [<subscrip_file_2>...]`

The **`-v`** option requests verbose output; more **`v`**'s (up to four) provide progressively more verbosity. The **`upp`** command has no direct output (unless verbosity is turned on), rather it mails a report of any actions taken to the email address specified in the subscription file.

There are no other command options for **`upp`**; its behavior is controlled entirely by the subscription file(s).

6.5 Automating UPP via cron

You can add a **cron** job that first sets up **UPD** then runs **UPP** with a subscription file (shown here as `upp.subscription`). Here is a sample **sh** script to which we give the filename `upp.launch`:

```
#! /bin/sh
. /usr/local/etc/setups.sh
setup upd
upp /path/to/upp.subscription
```

A sample crontab entry to run the `upp.launch` script every night at midnight might look like:

```
0 0 * * * /path/to/upp.launch
```

Chapter 7: Installing Products using FTP

This chapter describes how to download a product using **FTP**, install it, and declare it to a local **UPS** database. Anonymous **FTP** is available on *fnkits*, and may be available on other **UPS** product distribution nodes. **FTP** does not take advantage of the local node's **UPD** configuration. It can be used only to retrieve products; it is left to the installer to unwind and declare them. Furthermore, if the table file and/or the `ups` directory is (are) not included the tar file, each must be retrieved separately.

FTP is not recommended for installations into the usual local product area; **UPD** is designed and configured specifically for this purpose and should be used instead. **FTP** is more suited to product installations into non-standard locations on your node, e.g., into your own area for use just by you.

On *fnkits*, **FTP** is most useful for off-site users who want to download FermiTools products, which are located under the `/pub` directory in the `KITS` database. You do not need to be a registered user to obtain the FermiTools products.

7.1 UPS Product Components to Download

One of the features of **UPS/UPD** v4 is that it allows product developers to update certain portions of a product without cutting an entire new release of the product.[1] Specifically, a developer can update any file within a product's `ups` directory and reissue the `ups` directory tar file, and/or update and reissue a product's table file independently of the product tar file. The disadvantage this feature presents is that you must download these elements separately when using **FTP** to install a product.

The files that are commonly found within a product's `ups` directory include:

- a `README` file which provides information about the product such as origin, developer, support level, and so on
- unformatted man pages (under `ups/toman/man`)
- formatted man pages (under `ups/toman/catman`)
- an `INSTALL_NOTE` file, when needed, with instructions for installers
- (sometimes) a table file[2]

1. In versions of **UPS/UPD** prior to v4, `KITS` contained one tar file per product. If anything in the product changed, it required adding a brand new tar file of the whole product to `KITS`.
2. Since the table file may get updated separately from the other `ups` directory files, the copy maintained in the `ups` directory is not always the most recent one.

7.2 Installing Products from fnkits.fnal.gov

First, verify that your node is registered to obtain products from *fnkits*. If not, complete the product distribution registration form at `http://www.fnal.gov/cd/forms/upd_registration.html`.

If you only want to access FermiTools products (which includes all products located under the `/pub` directory), registration is not required.

The naming conventions and file hierarchy on *fnkits* have been constructed to make finding and downloading product files relatively easy. We show the procedure by way of an example, using the (fictional) product **sister**, version v1_0, for flavor Linux+2. For the local database we use `/fnal/ups/db` and we take the local product area to be `/fnal/ups/products`.

7.2.1 Download the Files from fnkits

In order to download the product files from the server, first change to an appropriate directory and run **FTP** to the machine, e.g.,:

```
% cd /usr/tmp
```

```
% ftp fnkits.fnal.gov
```

Provide the username *anonymous*, and use your <*username*>@<*nodename*> as the password.

Once you're logged on, you need to find the product you want. If you know the product's name, version, and flavor, you can just **`cd`** to the appropriate directory. If not, you may need to browse a bit. The product pathnames are listed in section 4.3.2 *Product Pathnames for FTP Access*. Products are arranged (via symlinks) in several different file hierarchies to make browsing easier:

- The `/products` directory contains products organized by product name and version.
- The `/KITS` hierarchy contains products organized by operating system.
- The `/pub` hierarchy contains the FermiTools products which are available to the general public.

We want to install the product **sister** version 1_0 for the flavor Linux+2, so we **`cd`** to the appropriate directory under `/products` and list the directory contents (this shows typical contents for products on *fnkits*):

```
ftp> cd /products/sister/v1_0/Linux+2
```

```
ftp> ls -l
```

```
drwxr-xr-x   4 100      kits         512 Sep 10 19:53 sister_v0_1_Linux+2
-rw-rw-r-x   1 100      kits        1538 Sep 10 19:53 sister_v0_1_Linux+2.table
-rw-r--r-x   1 100      kits     9687040 Aug 19 21:05 sister_v0_1_Linux+2.tar
-rw-rw-r-x   1 100      kits       60928 Sep 10 19:53 sister_v0_1_Linux+2.ups.tar
```

The directory `sister_v0_1_Linux+2` contains the unwound `ups` directory files (to allow you to browse, read and/or download individually any of the files it contains). `sister_v0_1_Linux+2.table` is the table file, `sister_v0_1_Linux+2.tar` is the complete product tar file, and `sister_v0_1_Linux+2.ups.tar` is a separate tar file of the `ups` directory.

Set the mode to “binary”, and **`get`** the two tar files:

```
ftp> binary
ftp> get sister_v0_1_Linux+2.tar
ftp> get sister_v0_1_Linux+2.ups.tar
```

Then set the mode to “ascii”, and get the table file:

```
ftp> ascii
ftp> get sister_v0_1_Linux+2.table
```

and exit:

```
ftp> bye
```

7.2.2 Unwind the Files into your Products Area

You need to unwind/copy the product files on your local node in the right order to ensure that:

- the individually-downloaded table file takes precedence over any previously existing table file as well as over one which may be contained within the product tar file
- the product's `ups.tar` file contents take precedence over any previously existing `ups` directory contents as well as over that which is contained within the product tar file.

This involves first unwinding the product tar file, then the `ups` directory, and finally copying the table file to its correct location. This procedure is illustrated below.

Note: From a technical standpoint, you are not required to follow any file naming/location conventions laid out in your system's `updconfig` file, if any, since you are not using **UPD** for the installation.

First make the product root directory:

```
% cd /fnal/ups/products
% mkdir -p sister/v0_1/Linux+2
```

Change to the product root directory and unwind the product tar file:

```
% cd sister/v0_1/Linux+2
% tar xvf /usr/tmp/sister_v0_1_Linux+2.tar
```

Now change to the product's `ups` directory (or make one if it doesn't exist) and unwind the product's `ups.tar` tar file:

```
% cd ups
% tar xvf /usr/tmp/sister_v0_1_Linux+2.ups.tar
```

Finally, change to the directory in which you want to put the table file and copy it in. Here we use the product directory under the database (the other commonly used location is under the product's `ups` directory).

```
% cd /fnal/ups/db/sister
% cp /usr/tmp/sister_v0_1_Linux+2.table ./sister.table
```

7.2.3 Declare the Product to your Database

You now need to declare the product instance to your **UPS** database[1]. Declaring a product instance is described in section 3.4 *Declaring an Instance Manually*.

To declare the downloaded product **sister** to our `/fnal/ups/db` database, we run the **`ups declare`** command as follows:

```
% ups declare sister v0_1 -f Linux+2 -z /fnal/ups/db \
  -r /fnal/ups/products/sister/v0_1/Linux+2 \
  -m sister.table
```

The **`-U`** and **`-M`** options are not included since we put the table file and `ups` directory in default locations where **UPS** will find them.

7.3 Installing Products from Other Product Distribution Nodes

The procedure for downloading from any standard **UPS** product distribution node is basically the same as illustrated for *fnkits* in section 7.2 *Installing Products from fnkits.fnal.gov*. The **UPD** configuration of the server node will most likely be different however, which means that the product and its associated files may be organized differently than on *fnkits*. You may need to verify that your node is registered to obtain products from the server. Contact the server maintainer or other designated person for information regarding node/user registration.

7.3.1 Locate the Product Files on the Server

The most reliable way to determine the location of the product files is to use the **`upd list`** command, e.g.,:

```
% upd list -h <hostname> -K+:@prod_dir:@ups_dir:@table_file  \
  sister v0_1 -f Linux+2
```

(We show the output on separate lines for readability:)

```
"/P/tar/sisterv0_1Linux+2.tar"
"/P/ups/sisterv0_1Linux+2.ups.tar"
"/P/table/sisterv0_1Linux+2.table"
```

In this example, files are organized on the server by type rather than by product:

- product tar files are stored under the `/P/tar` hierarchy
- product `ups` directory tar files are stored under the `/P/ups` hierarchy
- table files are stored under the `/P/table` directory.

1. ... unless you're not running **UPS** on your local node.

7.3.2 Download the Files from the Server

Let's take *special.upd.host* as our server node. In order to download the product files from the server, first change to an appropriate directory and run **FTP** to the machine, e.g.,:

```
% cd /usr/tmp
% ftp special.upd.host
```

Provide the username *anonymous*, and use your <*username*>@<*nodename*> as the password.

Once you're logged on, set the mode to "binary", and **`get`** the two tar files:

```
ftp> binary
ftp> cd /P/tar
ftp> get sister_v0_1_Linux+2.tar
ftp> cd /P/ups
ftp> get sister_v0_1_Linux+2.ups.tar
```

Then set the mode to "ascii", and get the table file:

```
ftp> ascii
ftp> cd /P/table
ftp> get sister_v0_1_Linux+2.table
```

and exit:

```
ftp> bye
```

7.3.3 Unwind the Files into your Products Area

Unwind the tar files and copy the table file as shown in section 7.2.2 *Unwind the Files into your Products Area*.

7.3.4 Declare the Product to your Database

Declare them as shown in section 7.2.3 *Declare the Product to your Database*.

Chapter 8: Product Installation: Special Cases

This chapter provides product installation information about specific cases. It discusses:

- how to install products requiring special privileges
- how to install into a local products area using the installation of **UPD** in AFS space
- how to install products into the AFS-space **UPS** products area

8.1 Installing Products that Require Special Privileges

Certain products supplied by the Computing Division need "special configuration" which can only be performed by a suitably privileged account. This is described in TN0092 *What does 'InstallAsRoot' Really Mean?*. The text here is adapted from that document.

For these particular products, listed below, at some point during the installation process you will be prompted to login as *root* and run the command[1]:

```
% ups installasroot <product> <version> [<options>]
```

This command would then proceed to run the privileged installation actions. The `INSTALL_NOTE` file should provide instructions for you if this is necessary.

Examples of products requiring configuration by a privileged account include:

python, **perl**	require that files and symlinks be created in `/usr/local/bin` for the convenience of users and system administrators (so that **perl** and **python** are always accessible, even if not previously setup).
tcsh, **bash**	require that files be copied to `/usr/local/bin` with proper permissions and ownership (for security reasons)
ssh	requires that configuration files and binaries be copied to system areas

1. By convention, the products' table files generally contain an installAsRoot action, which gets executed via this command. For particularly complicated products, the installAsRoot action my point to other developer-defined actions, and you may be instructed to run one or more customized commands instead of **`ups installasroot`**.

kerberos requires that configuration files and binaries be copied to system areas; also requires **suid** on certain files under the product area $KERBEROS_DIR itself.

systools requires that **suid** permissions be set on various **cmd** plug-in scripts under the $PRODUCTS area.

On many systems, `/usr/local` and/or the $PRODUCTS area are NFS-mounted. For security reasons, these areas may not in fact be writable by the *root* account on the node where the product installation is taking place.

Note that in AFS file systems, *root* access is usually insufficient to guarantee write access. At present, however, there are no products known to require an admin token for their installAsRoot actions.

If you are instructed to issue a special installation command, e.g.,:

```
% ups installAsRoot <product> <version>
```

assume that you need full write access to the following locations:

`/usr/local` Scripting languages, local utilities, and certain security tools will require symlinks and/or files under `/usr/local/bin` (or `/usr/local/etc`). Bear in mind that in a mixed-platform cluster, `/usr/local` will typically comprise a set of directories, one for each type of system.

$PRODUCTS More accurately, *root* may need to write/modify configuration and/or log files under the area where products are installed. This is determined by the system's **UPD** configuration, usually found in the file `$PRODUCTS/.updfiles/updconfig`.

local system disk Security tools, system administration tools, web servers, and so on, may need to write configuration files into system areas such as `/etc` and `/var`.

If access to other areas is required, it should be noted in the product's `INSTALL_NOTE` file. The steps to take in order to ensure that areas listed above are writable will vary depending on the particular configuration of each system, and are left to the system administrator.

For some older versions of products, a symbolic link gets created in `/usr/local/products` whenever a new instance is declared to the database. These products will need to be configured on each machine with a unique `/usr/local` area. This packaging philosophy has been phased out.

8.2 Installing Locally Using UPD from AFS-Space

Systems running AFS can be configured to provide access both to locally installed/declared products and to products in AFS space without maintaining **UPS/UPD** in the local database. This configuration is described in section 12.2 *Configuring a Local Database to Work With AFS*.

The local database is usually given a standard name in common use at Fermilab:

`/fnal/ups/db`	standard for several product server bootstrap configurations
`/local/ups/`	standard for Fermi RedHat Linux
`/usr/products/`	another popular naming convention
`/usr/products/CMSUN1/`	naming convention for CMS local databases

The database must point to a local **UPD** configuration file with appropriate product location definitions.

With no locally installed **UPS/UPD**, you'll need to invoke the AFS installation of **UPD** to install a product into the local products area. If the first or only database listed in $PRODUCTS that contains **UPD** is the AFS database, then you need no database specifier in the **`setup`** command. Run it normally:

```
% setup upd
```

Assuming that $PRODUCTS lists your local database first, you can run **`upd install`** without any database option and your product will go into the local database (assuming the **UPD** configuration is set accordingly). If $PRODUCTS doesn't list yours first, include the **`-z`** option in the **`upd install`** command, e.g.,:

```
% upd install -z /path/to/local/db <product> <version> ...
```

In terms of where they get installed and declared, dependencies are treated the same as the main product. If you want to install and declare only the main product locally (e.g., for development) but you want to keep all of its dependencies in AFS space, use the syntax:

```
% upd install -j [-z /path/to/local/db] <product> <version> ...
```

to install only the main product. Then, as needed, install the dependencies in AFS-space; see section 8.3 *Installing Products into AFS Space*.

8.3 Installing Products into AFS Space

8.3.1 Overview

A single AFS *volume* is intended to contain instances for all flavors of a particular **UPS** product-version combination. For each product, there is a read/write volume into which the product must be installed:

`/afs/.fnal.gov/ups/<product>/<version>`

(note the dot preceding `fnal`). There is a process called *releasing a volume* which replicates the read/write product volume into *read-only clones*. Replication avoids any single point of failure for a product and provides more robust service. Multiple frozen read-only copies of the product areas are kept on several AFS server machines. We want users running **`setup`** to access these redundant, read-only volumes of products. Otherwise, the benefit of cloning is wasted. A product must therefore be declared to **UPS** via its read-only pathname (notice the absence of a dot preceding `fnal`):

`/afs/fnal.gov/ups/<product>/<version>`

The AFS-space `updconfig` file is configured to unwind products into the read/write area (via the UNWIND_PROD_DIR definition), and then release them to the read-only volumes, using the `upd_volrelease` script (via the PREDECLARE action). As the action name suggests, this happens before declaring the product. When **`ups declare`** gets called, the PROD_DIR_PREFIX in the AFS-space `dbconfig` file ensures that the read-only pathname gets declared.

Installations into AFS space should be made from one of the interactive nodes of the *fnalu* cluster, preferably a SunOS node.[1] The *fnalu* nodes have the **arcd** daemon running and supply the `upd_volrelease` script, both of which are required for AFS installations. *fsui02.fnal.gov* is the machine on which it works most consistently. If you need to use a non-SunOS node, use **`upd install -H <target_flavor>`** to set the flavor. The userid *products* and the groups *uas* and *upsdatabase* are allowed to install into AFS space.

8.3.2 Request a Product Volume

Only AFS administrators are allowed to create product volumes. To request a product volume, contact Customer Support at *helpdesk@fnal.gov.* Customer Support will forward the message appropriately. Recall that all the instances for the different flavors of a particular product-version pair go into a single AFS volume. Your request needs to include:

- the product name
- the product version
- the combined size of the various instances that will go into the volume
- the AFS user and group(s) who need write access

An AFS administrator will create a volume writable by you (the requestor) and your group, and notify you when it's ready. The product instances can be installed in the volume as soon as it is created.

8.3.3 Install your Product

Install each instance of your product (all of the same version) using the **`upd install`** command (documented in Chapter 5: *Installing Products Using UPD*). Specify the read-only path for the database as shown here:

```
% upd install -z /afs/fnal.gov/ups/db <product> <version> \
  -f <flavor> [-q <qualifierList>] ...
```

1. On some AFS nodes the `upd_volrelease` script is missing from `/usr/local/bin`, and on others the setup work for it has not been completed.

8.3.4 Post-Installation Steps

Configure/Tailor the Product

Because the product areas as declared in AFS space are read-only, if your product requires configuration or tailoring, you must execute these commands using the read/write path name, e.g.,:

```
% ups configure -r /afs/.fnal.gov/... <product> <version> \
  -f <flavor>...
```

```
% ups tailor -r /afs/.fnal.gov/... <product> <version> \
  -f <flavor> ...
```

For some products, notably **perl** and **python**, the configure script/action checks for `/afs/` and makes the appropriate path change, e.g.,:

```
DEST_DIR=`echo $UPS_PROD_DIR | sed -e 's;/afs/fnal.gov;/afs/.fnal.gov;'`
```

Create Symbolic Links

If the product needs the ability to write into any areas under its product root directory during normal use, then you need to symbolically link these areas. If the area must be shared, link to the read/write area. If not, you can link to some area on non-AFS writable disk (e.g., under `/tmp` or `/var`).

For example, say the product **fred** in AFS space needs to write into `$FRED_DIR/log` which is read-only. Go into the read/write $FRED_DIR area, remove the log directory and create a link for the area in which to write (e.g., `/var/tmp/fred`) in the read-write area, e.g.,:

```
% ln -s /var/tmp/fred /afs/.fnal.gov/ups/fred/v1_0/SunOS/log
```

You will then need to release the volume, as described below. Your read-only replicas will contain the link.

If links are made to a non-AFS writable disk, check the SETUP action in the product's table file; it should ensure that the specified area exists at product setup. E.g., if linking to `/var/tmp/fred`:

```
Action = setup
...
Execute( test -d /var/tmp/fred || (mkdir /var/tmp/fred;
         chmod 777 /var/tmp/fred), NO_UPS_ENV) (all on one line in real file)
```

Rerun the Volume Release

If you have configured and/or tailored the product, or if you have added symbolic links, you need to manually rerun the `upd_volrelease` script to re-release the product volume, e.g.,:

```
$ upd_volrelease /afs/.fnal.gov/ups/<product>/<version>
```

unless the product's actions already take this into account (look for `upd_volrelease` calls in the table file's CONFIGURE action).

Note that it doesn't hurt to re-release a product volume several times in a row, so if you're not sure, just rerun it.

To save time, configure and/or tailor all the flavors of your product version first, and then run the **`upd_volrelease`** command once at the end.

Chapter 9: Troubleshooting UPS Product Installations

This chapter provides a few hints if things don't seem to work after installing a product.

- If you don't find a product that you expect to see on the **FTP** server, it could be that the product is flagged as belonging to a special category to which you don't have access (e.g., site-only and U.S.-only are two of the categories used on *fnkits*; see section 21.3.2 *The Recognized Product Categories*). You may need to try with a different userid. It is also possible, if not terribly likely, that the file's permissions are set incorrectly on the server.
- If the $PATH goes away, restore it by running:

  ```
  % setup setpath
  ```

  and check if the **pathSet** function is used in the table file -- if it is set wrong, this may be the cause.
- To print out diagnostic information about what might be wrong with the installation, run **ups verify**:

  ```
  % ups verify -a <product> [<version>]
  ```
- Try setting up just the main product and none of its dependencies. This should help determine which file has the problem, the main one or a dependency. Use **-j** in the **setup** command:

  ```
  % setup -j <product>
  ```
- Print out verbose information using the **-v** option with **setup**:

  ```
  % setup -v <product>
  ```

  To get progressively more information, use multiple **v**'s, e.g., **-vv**, **-vvv** (up to four).
- Check file permissions. Any scripts called by the table file must be both readable and executable. The product executable(s) must of course be executable. The product database files must be readable.
- To examine the temporary file that the **setup** command creates and sources, run the command:

  ```
  % ups setup <product> [<version>]
  ```

  This returns the path of this temporary file, and you can then go look at the file. For example:

  ```
  % ups setup ocs
    /var/tmp/aaaa00273
  ```
- For most **UPS** commands, the **-s** option can be used to simulate the command (i.e., create the temporary file) without executing it. It also returns the path of the temporary file it created, for example:

  ```
  % setup -s -z /products/ups_database/upsII/main xpdf
  INFORMATIONAL: Name of created temp file is /var/tmp/aaaa005Mt
  ```

- If home directories move or if older versions of products have been deleted, you might want to prevent execution of unsetup files prior to a subsequent setup. In this case, don't **`unsetup`** the product. Just setup the product again using **`-k`**:

```
% setup -k <product>
```

Part III System Administrator's Guide

Chapter 10: *Maintaining a UPS Database*

In this chapter we assume that you have **UPS/UPD** installed and that you have a working database and products area. We provide instructions and examples for performing the following functions:

- declaring product instances to a database
- declaring, removing and changing chains
- removing product instances
- verifying the integrity of a product instance
- modifying information in a database file
- determining if a product needs to be updated
- updating a table file or `ups` directory
- retrieving an individual file from a distribution node
- checking product accessibility
- troubleshooting

Chapter 11: *UPS and UPD Pre-install Issues and General Administration*

In this chapter we take a step back with regard to Chapter 10: *Maintaining a UPS Database*, and assume that you have not yet installed **UPS/UPD**, or created a **UPS** database and products area. We guide you through the administrative decisions and tasks that are involved in preparing to implement **UPS/UPD**. Towards the end of the chapter there is also some information regarding general administrative tasks. For machines running AFS or NFS, also see Chapter 12: *Providing Access to AFS Products*.

Chapter 12: *Providing Access to AFS Products*

This chapter describes how to provide access on your local machine to **UPS** products installed in AFS space.

Chapter 13: *Bootstrapping CoreFUE*

CoreFUE is a bundled product which includes **UPS/UPD** and **perl**. It refers to the core components of the Fermi UNIX Environment (FUE). When we discuss installing **UPS/UPD**, we're generally talking about **coreFUE** since **perl** is a required component. Here we describe how to use automated

scripts to bootstrap **coreFUE**, that is, to install **coreFUE** on a machine on which no prior versions of these products are installed. Several project-specific configurations of **UPS**/**UPD** are available.

Chapter 14: *Automatic UPS Product Startup and Shutdown*

This chapter covers configuring your system to support automatic startup and shutdown of **UPS** products, and installing individual **UPS** product instances to start and stop automatically.

Chapter 10: Maintaining a UPS Database

In this chapter we assume that you have **UPS/UPD** installed and that you have a working database and products area. We provide instructions and examples for performing the following functions:

- declaring product instances to a database
- declaring, removing and changing chains
- removing product instances
- verifying the integrity of a product instance
- modifying information in a database file
- determining if a product needs to be updated
- updating a table file or `ups` directory
- retrieving an individual file from a distribution node
- checking product accessibility
- troubleshooting

To get command usage information or on-line help, use the following resources:

- Refer to Part VI of this guide (in GU0014A), *Command Reference*, especially Chapter 22: *UPS Command Reference*.
- Run the command with **`"-?"`**, e.g., **`ups declare "-?"`**[1].
- Man pages are also provided; use an underscore with the **UPS** command when running **`man`**, e.g., **`man ups_declare`**.

10.1 Declare an Instance

A product instance must exist on the system before it can be declared to a **UPS** database[2]. Product declaration is done with the **`ups declare`** command. Declaring a product instance makes it known to **UPS**, and therefore retrievable within the **UPS** framework. Normally products are installed on user nodes using the **`upd install`** command which, in addition to downloading and installing the product, runs **`ups declare`** to make the initial declaration of the product to the local **UPS** database. If you use **FTP** to download a product, then you'll need to declare it manually. Refer to Chapter 7: *Installing Products using FTP* for details about installing with **FTP**.

1. The double quotes are necessary for C shell users; **`-?`** is interpreted by **sh**.
2. At least a rudimentary root directory hierarchy for the product, its table file directory and table file must exist before declaration.

If you use **`upd install`** and you have more than one database, refer to section 5.2 *How UPD Selects the Database* to see how **UPD** determines the database for the declaration.

10.1.1 The ups declare Command

Before declaring, make sure the product is unwound into in its final location. Also make sure that you've downloaded the table file and installed it in an appropriate directory. For an initial declaration you must specify at a minimum: the product name, product version, product root directory, flavor and table file name[1].

The full command description and option list is in the reference section 22.5 *ups declare*. Here we show commonly used command options (see the notes regarding **`-z`**, **`-U`** and **`-M`** which follow):

```
% ups declare <product> <version> -r /path/to/prod/root/dir/ \
  -f <flavor> [-z /path/to/database] [-U /path/to/ups/dir] \
  [-m <table_name>.table] [-M /path/to/table/file/dir] \
  [<chainFlag>]
```

1) If the database is not specified using **`-z`**, **UPS** declares the product into the first listed database in $PRODUCTS (see section 26.1 *Database Selection Algorithm* for more information).

2) If the product's `ups` directory tar file was unwound in the default location (`$<PRODUCT>_DIR/ups`), then **`-U /path/to/ups/dir`** is not needed. If the `ups` directory is located elsewhere (or named differently), this specification must be included. If specified as a *relative* path, it is taken as relative to the product root directory.

3) If the product's table file was placed in either of the two default locations (under `/path/to/database/<product>/` or in the product's `ups` directory), then **`-M /path/to/table/file/dir`** is not needed. Only use the **`-M`** option if you have moved the table file to a separate location where **UPS** won't otherwise find it. If specified as a *relative* path, it is taken as relative to the product root directory. See section 28.4 *Determination of ups Directory and Table File Locations* for details on how **UPS** finds the table file.

Unless the product you're declaring has no table file (true for very few products), make sure its location gets declared properly, either explicitly or by default. Otherwise, users will need to specify its name and location on the command line every time they want to run or operate on the product. If it is neither declared nor specified on the command line, **UPS**/**UPD** assumes there is no table file.

You can opt to declare a chain to the product instance at this time or in a later declaration. To declare a chain, include the appropriate chain flag in the command (see section 1.3.5 *Chains* for a listing). This is described in section 10.2 *Declare a Chain*.

10.1.2 Examples

Additional examples are included in the reference section 22.5 *ups declare*.

1. Two exceptions: (1) if the product consists only of a table file that sets up a list of dependencies, there is no product root directory; and (2) if the product has no table file (very rare) then there is no table file name.

Declaration of New Product to Non-default Database

The following command shows a fairly typical product declaration. We'll install a product called **histo** v4_0 onto a SunOS+5 node. We assume the product instance's `ups` directory is maintained under its product root directory, and that it contains the table file. We include the **`-z`** option to indicate that we want to override the default database selection. This is the first instance of this product to be declared to this database, therefore the **`ups declare`** command automatically creates the appropriate product directory under the specified database:

```
% ups declare histo v4_0 -f SunOS+5 -m histo.table -z $MY_DB -r\
  /path/to/products/SunOS+5/histo/v4_0
```

We can run a **`ups list -l`** command to see all the declaration information (include **`-a`** because it's not yet declared current):

```
% ups list -alz $MY_DB histo
```

```
DATABASE=/path/to/ups_database/declared
       Product=histo   Version=v4_0    Flavor=SunOS+5
              Qualifiers=""   Chain=""
              Declared="1998-04-17 22.08.30 GMT"
              Declarer="aheavey"
              Modified="1998-04-17 22.08.30 GMT"
              Modifier="aheavey"
              Home=/path/to/products/SunOS+5/histo/v4_0
              No Compile Directive
              Authorized, Nodes=*
              UPS_Dir="ups"
              Table_Dir=""
              Table_File="v4_0.table"
              Archive_File=""
              Description=""
              Action=setup
                     prodDir()
                     setupEnv()
                     addalias(histo,${UPS_PROD_DIR}/bin/histo)
                     addalias(hsdir,${UPS_PROD_DIR}/bin/hsdir)
                     envSet(HISTO_INC,${UPS_PROD_DIR}/include)
```

Declaration of Additional Instance of a Product

In the following example we declare an additional instance of **histo**, of the same version, but for the flavor IRIX+5. Again the table file resides under the product root directory's `ups` subdirectory, and we override the default database. This time we declare it with the chain "test" (**`-t`**):

```
% ups declare histo v4_0 -tf IRIX+5 -m histo.table -z $MY_DB -r\
  /path/to/products/IRIX+5/histo/v4_0
```

Running a **ups list -a** to see what the database now contains for this product, we find:

```
% ups list -az $MY_DB histo
```

```
DATABASE=/path/to/ups_database/declared
        Product=histo   Version=v4_0    Flavor=SunOS+5
                Qualifiers=""   Chain=""

        Product=histo   Version=v4_0    Flavor=IRIX+5
                Qualifiers=""   Chain=test
```

Declaration with Table File Located in Database

Depending on your configuration, you may want the table file to reside in the product's subdirectory under the database (e.g., `$PRODUCTS/<product>/<table_file>`).

A table file for the product must be placed in its permanent location before the instance is declared to the database. Therefore, if you are declaring the first instance of a product to the database, you need to manually create the product directory under the database and copy the table file into it before declaring the instance.

You still do not need to specify the table file location (**-M** option) on the **ups declare** command line; **UPS** will find it here.

10.2 Declare a Chain

Chains are described briefly in section 1.3.5 *Chains*, and in detail in Chapter 29: *Chain Files*. A chain can be declared when the product instance is initially declared to the database (see section 10.1 *Declare an Instance*), or at a later time.

10.2.1 The ups declare Command with Chain Specification

To add a chain to a product instance, use the **ups declare** command with a **chainFlag** option. The **chainFlag** option can be one of the standard ones: **-c**, **-d**, **-n**, **-o**, or **-t**. **chainFlag** can also be replaced by **-g chainName**, where **chainName** is either one of the standard chain names, e.g., **-g current**, or a user-defined one. The full command description and option list is in section 22.5 *ups declare*. Here are some examples:

```
% ups declare -c [<other options>] <product> <version>
```

```
% ups declare -g current [<other options>] <product> <version>
```

```
% ups declare -g my_chain [<other options>] <product> <version>
```

Declaring a chain is generally allowed on any node of a cluster, however if the corresponding chain action in the table file includes any node-specific or flavor-specific functions,[1] we strongly recommend that you declare the chain from that node, or from a node of that flavor to avoid mismatches. This should be noted in the `INSTALL_NOTE` file if it's necessary.

1. Actions are described in Chapter 33: *Actions and ACTION Keyword Values*, functions in Chapter 34: *Functions used in Actions*, and table files in Chapter 35: *Table Files*.

To include a chain in the initial declaration, simply add a chain option to the instance declaration as described in section 10.1 *Declare an Instance*. To add a chain to a previously declared product instance, include only the options required to identify the product instance and the chain option, e.g.,:

```
% ups declare -c <product> <version> [-f <flavor>] \
  [-z <database>]
```

In general, this does not change any existing chain, it adds a new one. However, if you have an instance already chained, and you wish to declare a new instance of a different version but the same flavor/qualifier pair to the same chain, the pre-existing chain will be removed automatically. In other words, **UPS** ensures that a chain for a particular flavor/qualifier pair is unique.

A couple of examples will help to clarify how this works. In these examples we assume that the product instance has previously been declared to the database either with no chain or with a different chain. Some of these commands will also work for declaring an instance initially to the database with a chain, however we refer you to section 10.1 *Declare an Instance* for examples specific to that operation.

10.2.2 Examples

Declare an Instance to the Database as test

In a typical situation, a product instance is initially declared as test (**`-t`**) to the default database, to be made current at a later date. In this example, we make an initial declaration as "test" of the product **histo** version v4_0, flavor IRIX+5, located in `/usr/products/IRIX+5/histo/v4_0`, with the table file name `v4_0.table`:

```
% ups declare -tr /usr/products/IRIX+5/histo/v4_0 -f IRIX+5 \
  -m v4_0.table histo v4_0
```

We verify the declaration using **`ups list -l -a`**:

```
% ups list -la histo -f IRIX+5
```

```
DATABASE=/path/to/ups_database/declared
       Product=histo   Version=v4_0    Flavor=IRIX+5
              Qualifiers=""   Chains=test
              Declared="1998-04-17 22.27.16 GMT:1998-04-17 22.27.16 GMT:1998-
              Declarer="aheavey:aheavey"
              Modified="1998-04-17 22.27.16 GMT:1998-04-17 22.27.16 GMT:1998-
              Modifier="aheavey:aheavey"
              Home=/path/to/products/IRIX+5/histo/0
              No Compile Directive
              Authorized, Nodes=*
              UPS_Dir="ups"
              Table_Dir=""
              Table_File="v4_0.table"
              Archive_File=""
              Description=""
              Action=setup
                     prodDir()
                     setupEnv()
                     addalias(histo,${UPS_PROD_DIR}/bin/histo)
                     addalias(hsdir,${UPS_PROD_DIR}/bin/hsdir)
                     envSet(HISTO_INC,${UPS_PROD_DIR}/include)
```

Notice that DECLARED, DECLARER, MODIFIED and MODIFIER all have two values. The first value is for the declaration to the database, the second is for the test chain declaration. In the following example, you will see that these fields acquire a third value when the chain is changed.

Change instance from test to current

Once testing is complete and successful, you will want to take the product instance out of test and declare it as current. For the product instance of the previous example, we issue the command:

```
% ups declare -c histo v4_0 -f IRIX+5
```

This adds the current chain, but it does not remove or modify the test chain. (To remove the test chain, see the instructions in section 10.3 *Remove a Chain*.) Verify using **`ups list`**:

```
% ups list -a histo -f IRIX+5
```

```
           Product=histo   Version=v4_0    Flavor=IRIX+5
                   Qualifiers=""   Chains=test,current
```

If we use the long form, we see the additional declaration and modification userid and time (output edited for brevity):

```
% ups list -la histo -f IRIX+5
```

```
DATABASE=/path/to/ups_database/declared
        Product=histo   Version=v4_0    Flavor=IRIX+5
                Qualifiers=""   Chains=test,current
                Declared="1998-04-17 22.27.16 GMT:1998-04-17 22.27.16 GMT:1998-04-18
22.00.16 GMT
                Declarer="aheavey:aheavey:aheavey"
                Modified="1998-04-17 22.27.16 GMT:1998-04-17 22.27.16 GMT:1998-04-18
 22.00.16 GMT
                Modifier="aheavey:aheavey:aheavey"
                ...
```

Change current Chain to Point to a New Instance

Another frequently encountered situation is that in which you already have a version chained to current and you want to declare a different version of the product as current for the same flavor. We'll use the previous example **histo** v4_0, and declare version v4_1 as current:

```
% ups declare -c histo v4_1 -f IRIX+5
```

The previously current instance for this flavor/qualifier pair now has no current chain. Any other chains it may have had (test, in this case) remain unchanged.

10.3 Remove a Chain

To remove a chain from a product instance, you can use the **`ups undeclare`** command, or you can simply remove the chain file, or the portion of it that relates to the instance in question. It is usually easier and less error-prone to use the **`ups undeclare`** command. The full command description and option list is in section 22.18 *ups undeclare*.

The **ups undeclare** command has a simple syntax for removing chains:

```
% ups undeclare <chainFlag> <product> [-f <flavor>] \
  [<other options>]
```

Do not include the version in the command; it is incompatible with including the chain, and may result in removing the product declaration! We recommend always including the **-f <flavor>** option if you have a multi-flavored database.

As an example, let's remove the current chain from the current instance of **ximagetools**. Running **ups list** before and after, we should see the current chain disappear:

```
% ups list -K+ ximagetools

    "ximagetools" "v4_0" "NULL" "" "current"

% ups undeclare -c ximagetools -f NULL

% ups list -aK+ ximagetools

    "ximagetools" "v4_0" "NULL" "" ""
```

If multiple flavor/qualifier pairs have the same chain and thus share the chain file in question (in which case you *must* specify the flavor/qualifier information on the command line), only the portion of the file relating to the specified instance will get removed; the file itself will not be deleted.

10.4 Change a Chain

In general, changing the chain to a product instance requires removing the pre-existing chain (see section 10.3 *Remove a Chain*) and adding a new one (see section 10.2 *Declare a Chain*). There is no way to directly change a chain.

When a current instance of a product already exists, if you declare a new instance of a different version but of the same flavor/qualifier pair as current, the `current.chain` file contents changes to point to the new version. This is true for any chain value, not just for current.

10.5 Undeclare and Remove an Instance

To undeclare a product instance means to remove all information pertaining to it from the **UPS** database in question. The information that gets removed includes:

- the version file, or the portion of the version file, that pertains to the instance
- any chain files, or the portions of any chain files, that pertain to the instance

The command **ups undeclare** is provided for this operation. You can opt to remove the actual product in the product instance's root directory, as well, by using either the **-y** or **-Y** option, as described in section 10.5.1 *Using ups undeclare to Remove a Product*. The **ups undeclare** command executes **ups unconfigure** by default (see section 3.6.1 *Configuring a Product*). The unconfigure process can be suppressed by using the **-C** option with **ups undeclare**, however normally you want this process to execute. The full **ups undeclare** command description and option list is in section 22.18 *ups undeclare*.

It is also possible to configure **UPP** to remove a product automatically. This is discussed in section 10.5.3 *Using UPP to Remove a Product*.

Before removing anything, you should find out if any other products have the product instance in question declared as a dependency.[1] If so, you may want to reconsider removing it. Removal of the product instance may affect the operation of its parent products.

10.5.1 Using ups undeclare to Remove a Product

To remove a product instance, you must specify the *version* of the instance, not its *chain*, in the **`ups undeclare`** command. Specifying the chain removes only that chain, not the instance itself.

Using `ups undeclare` is the recommended procedure for removing product instances. Removing them manually does not ensure that all the files get deleted or that chains get updated properly, which can lead to a fragmented products area.

If you choose to completely remove the product, and you want to delete the product instance's directory tree starting from its root directory, use one of the options **`-y`** or **`-Y`** with **`ups undeclare`** (**`-y`** queries you for confirmation, **`-Y`** does not). We recommend always including the **`-f`** option if you have a multi-flavor database. You may also need to include the **`-z`** option if you have more than one database. The command syntax is (showing commonly used options):

```
% ups undeclare [-f <flavor>] [-q <qualifierList>] [-y|Y] \
  [-z <database>] <product> <version>
```

Special case: If a product has a CONFIGURE action that modifies files outside of its product root directory, and if this instance is used by more than one node, flavor or file system, then you may need to run **`ups undeclare`** or **`ups unconfigure`** on all of the nodes before removing the product files on any node. The `INSTALL_NOTE` file should indicate if this is the case. If you're not sure, check in the product's table file.

Example 1

In this first example, we remove the product **tcl** v7_6a. We undeclare it and opt to remove the product root directory after query, taking a "snapshot" before and after. First, verify the declared instances of **tcl** in the database:

```
% ups list -aK+ tcl
```

```
"tcl" "v8_0_2" "IRIX+5" "" "current"
"tcl" "v7_6a" "IRIX+5" "" ""
```

Next verify the product root directory contents (run **`setup`** to set $TCL_DIR, check contents of the **tcl** products area, and then list contents of $TCL_DIR):

```
% setup tcl v7_6a
```

```
% cd $TCL_DIR/../ ; ls -l
```

```
total 8
drwxrwxr-x    9 aheavey  g020         140 Sep 15 15:29 v7_6a/
drwxrwxr-x    9 aheavey  g020        4096 Sep  8 15:50 v8_0_2/
```

---

1. The **`ups parent`** command will provide this information. The command is not available as of **UPS** version v4_5_2; it is planned for a future release.

```
% cd v7_6a ; ls -l
```

```
total 40
-rw-r--r--    1 aheavey  g020         165 May  1  1997 BUILD_INFO
-rw-r--r--    1 aheavey  g020        5861 May  1  1997 Makefile
drwxrwxr-x    2 aheavey  g020          83 Sep 15 15:29 alt-ups
drwxrwxr-x    2 aheavey  g020          40 Sep 15 15:29 bin
...
```

Now undeclare **tcl** v7_6a and remove its product root directory structure. The **`-y`** option queries before removing, and we respond "**y**" for yes (one would enter "**n**" for no):

```
% ups undeclare -f IRIX+5 tcl v7_6a -y
```

```
Product home directory -
       /export/home/t1/aheavey/upsII/products/tcl/v7_6a/
Delete this directory?y
```

Once it finishes, verify the deletion:

```
% ups list -aK+ tcl
```

```
"tcl" "v8_0_2" "IRIX+5" "" "current"
```

```
% cd $TCL_DIR/../ ; ls -l
```

```
total 8
drwxrwxr-x    9 aheavey  g020        4096 Sep  8 15:50 v8_0_2
```

We see that the declaration was removed and the `v7_6a` directory is gone from the **tcl** product area.

Example 2

The following command is a dangerous example! We include it as a caution. It finds the best flavor match using the standard instance selection algorithm (see section 26.2 *Instance Matching within Selected Database*) and removes that instance of the product **pine** version v3_91 and any chains that point to it. It also removes the product root directory for this instance of **pine**; it does not query for confirmation before doing so.

```
% ups undeclare -Y pine v3_91
```

Depending on the instances you have in your database, you may end up removing the instance for, say, OSF1+V3 when you really wanted to remove the one for OSF1!

10.5.2 Undoing Configuration Steps

There is a **`ups unconfigure`** command for undoing configuration steps, described in section 22.17 *ups unconfigure*. Normally this command does not need to be run explicitly; the **`ups undeclare`** command undoes the reversible configuration operations by default.[1] Refer to the `INSTALL_NOTE` file for instructions.

1. When a product is undeclared, any steps in the table file under ACTION=UNCONFIGURE get executed by default, or the (reversible) functions under ACTION=CONFIGURE get undone. These concepts are explained in section 33.2.2 *"Uncommands" as Keyword Values*.

10.5.3 Using UPP to Remove a Product

It is possible to configure **UPP** to remove a product automatically. To do this you must create or edit a subscription file for **UPP**, which is documented in Chapter 32: *The UPP Subscription File*. Within the file you identify the instance(s), set a condition to trigger its removal, and provide the instruction to remove it.

There are two conditions that **UPP** recognizes:

- a new version of the product is available on the distribution node
- the chain on the specified product instance changes

The appropriate instruction to use for removing a product is `delete`, as documented in section 32.2.2 *Conditions and Instructions*. When the condition is met, **UPP** executes **`ups undeclare -Y`** for you (which removes the product root directory structure in addition to the declaration).

Here we provide a sample subscription file stanza for removing a product when its current chain gets removed on the server (we include the **`notify`** function in addition to **`delete`**, which is always a good idea):

`product = exmh`	Identify subscribed product as **exmh** (the **exmh** versions remain unspecified in this example, therefore act on all versions for the flavor specified below).
`flavor = SunOS+5.5`	Identify flavor of product (this is optional)
`action = uncurrent`	List in the following lines one or more functions to perform when an instance of the listed product-flavor combination is unchained from current on the server.
`notify`	Send a notification message to *<userid>@fnal.gov* (specified in file header)
`delete`	Remove the instance (declaration and product root directory) from the local node.

10.6 Verify Integrity of an Instance

The **`ups verify`** command checks the information in all the database files for the specified instance in order to determine if there are any errors or inconsistencies. The full command description and option list is in section 22.19 *ups verify*. Shown here with some commonly-used options, the command syntax is:

```
% ups verify -a <chainFlag> [-f <flavor>] <product> \
  [<version>]
```

Here is sample output for a product for which several files and directories listed in the version file were not found (**-a** is included to match all instances):

```
% ups verify -a blt
```

```
DATABASE=/path/to/upsdb
WARNING: File not found - /myman/
WARNING: File not found - /mycatman
WARNING: File not found - /myinfo/
WARNING: File not found - /myhtml/
WARNING: File not found - /mynews/
WARNING: File not found - /path/to/upsdb/.updfiles
INFORMATIONAL: Verifying product 'blt'
WARNING: File not found - /usr/products/IRIX/blt/v2_1
WARNING: File not found - /usr/products/IRIX/blt/v2_1/ups
```

10.7 Modify Information in a Database File

The **ups modify** command allows you to manually edit any of the database product files. It runs **ups verify** on the instance to perform syntax and content validation before and after the editing session. The full command description and option list is in section 22.12 *ups modify*. The command syntax with some commonly used options is:

```
% ups modify <product> [<version>] [-E <editor>] [<chainFlag>]\
  [-N <fileName>] [-z <database>]
```

ups modify performs the following steps (if you specify the file using **-N**, the menu will not appear):

- presents menu of files that you can edit and asks you to either select one or quit
- verifies pre-modification contents of file (runs **ups verify**)
- starts up the editor given by **-E <editor>** or, if that is not specified, then $EDITOR, if set. If neither is specified, it starts up **vi** by default.
- makes a copy of the file to be edited
- pulls copy of file into the editor
- after user exits the editor, runs **ups verify** on the edited file
- if the validation succeeds, writes the new file over the old one and quits
- if the validation does not succeed, provides informational messages, asks if you want to save changes, and quits
- if no changes made to file, again presents menu of files

Sample Session with (1) Unsuccessful and (2) Successful Validation

```
% ups modify teledata v1_0 -N $MYDB/teledata/v1_0.version
```

In this example, we select the version file (via **-N**) for the product **teledata** v1_0 (default flavor, no qualifiers). Since **-E** is not given, **UPS** will use the editor set in $EDITOR, or **vi** if that variable is not set. First, **UPS** runs **ups verify** and produces the output:

```
Pre modification verification pass complete.
```

No errors were detected. The version file is next displayed in the editor.

1) To illustrate an unsuccessful validation, we add a bogus line:

```
TESTKEYWORD = value
```

and save and quit. **UPS** returns the following messages, and we opt to save the erroneous change:

```
INFORMATIONAL: Unexpected key word 'TESTKEYWORD' in '/home/t1/aheavey/upsII/decl
ared/teledata/v1_0.version', line 17
INFORMATIONAL: Unexpected key word 'TESTKEYWORD' in '/home/t1/aheavey/upsII/decl
ared/teledata/v1_0.version', line 17
Post modification verification pass complete.
Do you wish to save this modification [y/n] ? y
```

UPS quits, saving the file as we requested.

2) To illustrate successful validation, we'll correct the error introduced above. We run the same **`ups modify`** command. **UPS** finds the error during the pre-edit validation:

```
INFORMATIONAL: Unexpected key word 'TESTKEYWORD' in '/home/t1/aheavey/upsII/decl
ared/teledata/v1_0.version', line 17
INFORMATIONAL: Unexpected key word 'TESTKEYWORD' in '/home/t1/aheavey/upsII/decl
ared/teledata/v1_0.version', line 17
Pre modification verification pass complete.
```

We remove the incorrect line from the version file, then save and quit. **UPS** displays the following message, and we elect to save the change (**y**):

```
Post modification verification pass complete.
Do you wish to save this modification [y/n] ? y
```

UPS quits, saving the file as requested.

Sample Session with No Changes

In this example, we select the current instance of the product **teledata**, and (by default) request a menu of files to edit:

```
% ups modify teledata
```

```
[0] /home/t1/aheavey/upsII/declared/teledata/current.chain
[1] /home/t1/aheavey/upsII/declared/teledata/v1_0.version
[2] /export/home/t1/aheavey/upsII/products/teledata/v1_0//ups/v1_0.table
[3] /home/t1/aheavey/upsII/declared/.upsfiles/dbconfig
Choose file to edit [0-3] or 'q' to quit: 1
Pre modification verification pass complete.
```

UPS starts up the editor and makes the selected file available to edit. We quit without making any changes. **UPS** displays the message:

```
No modifications, nothing to save.
```

UPS then displays the menu again, and we opt to quit:

```
[0] /home/t1/aheavey/upsII/declared/teledata/current.chain
[1] /home/t1/aheavey/upsII/declared/teledata/v1_0.version
[2] /export/home/t1/aheavey/upsII/products/teledata/v1_0//ups/v1_0.table
[3] /home/t1/aheavey/upsII/declared/.upsfiles/dbconfig
Choose file to edit [0-3] or 'q' to quit: q
```

10.8 Determine If a Product Needs to be Updated

UPP can be configured on a local machine to alert users via email when a newer version of a product is available in `KITS`, or when a product instance's table file or `ups` directory needs to be updated. If your installation is not configured to do this, you can use **UPD** interactively to find this information.

10.8.1 Using UPP

UPP can be used for several functions as described briefly in section 1.1 *Introduction to UPS, UPD and UPP*, and in detail in Chapter 32: *The UPP Subscription File*. For instructions on how to configure **UPP** to notify you regarding a product, see Chapter 32 or 10.5.3 *Using UPP to Remove a Product*.

10.8.2 Using UPD

To determine if you need to reinstall a product, use the **`upd install`** command with the **`-s`** option, as shown, while logged on to a node of the flavor you wish to check (or use the **`-H`** option to specify a different flavor). The full command description and option list is in the reference section 23.8 *upd install*.

```
% upd install -sv <product> [<version>] [-h <host>] \
  [-H <flavor>]
```

If it's ok, you'll see no output. If there's a discrepancy between what's on your node and what's on *fnkits* (or on the host specified using **`-h`**), you'll see output of the form:

```
Installing <product>
I would make directory /path/to/<product>/<flavor>/<version>
I would fetch directory <kitsflavor> from
ftp://fnkits.fnal.gov//ftp/products/<product>/<version> as
/path/to/<product>/<flavor>/<version> now
...
```

Although it says "Installing", it's only telling you what it would have to do in order to install.

If you are interested in knowing only if the product's table file or `ups` directory has been changed on the server and needs an update on your machine, use the **`upd update -s`** command. It compares the MODIFIED dates in the remote and local nodes. The full command description and option list is in the reference section 23.12 *upd update*.

```
% upd update -s <product> [<version>] <component> \
  [-H <flavor>]
```

The argument **`<component>`** can take the value **`table_file`** or **`ups_dir`**, or both, colon-separated. If no update is needed, there is no output. If an update is needed, the messages will inform you.

10.9 Update a Table File or ups Directory

The **upd update** command is used to update a product's table file and/or ups directory. It operates on the specified product instance and its dependencies by default. It retrieves the specified components from a distribution node and downloads them to the local node, overwriting the corresponding pre-existing component(s). The full command description and option list are in section 23.12 *upd update*. The command syntax with some commonly used options is:

```
% upd update <product> [<version>] <componentList>     \
  [-H <flavor>] [<chainFlag>] [-h <host>] [-i] [-j]
```

In the following example, we overwrite the table file for the product instance **xntp** v3_4, flavor SunOS. This operation will succeed if the MODIFIED date in the remote version file that points to the table file on the distribution node is later than that in the comparable local version file; no overwrite will occur otherwise. Before running **upd update**, we compare the MODIFIED dates for the product by using a **ups list** command like the following:

```
% ups list -f SunOS -K MODIFIED xntp v3_4

    ":1998-04-01 20.08.02 GMT"
```

on the local node, and running **upd list** with similar options on the distribution node (the default *fnkits* is used here):

```
 % upd list -H SunOS -K MODIFIED xntp v3_4

    "1998-09-10 08.13.07 GMT"
```

The MODIFIED date in the remote version file is later than that in the local version file, therefore we expect an update to occur.

Now we run the **ups update** command requesting the component **table_file**:

```
% upd update table_file xntp v3_4 -H SunOS

    updcmd::updcmd_update - Updating xntp.
    upderr::upderr_syslog - successful transfer
    ftp://fnkits.fnal.gov///ftp/upsdb/xntp/v3_4SunOS.table -> /tmp/mwmdb/xntp/v3_4.table
    upderr::upderr_syslog - successful ups touch xntp v3_4 -f SunOS -q "" -U ""
```

Rerun the **ups list** command to verify that the MODIFIED date changed, indicating that the update took place.

To update several instances of **xntp** v3_4 for a list of flavors, use the **-H** option like this:

```
% upd update table_file xntp v3_4 -H SunOS:IRIX:OSF1:Linux
```

Using **-H** ensures that all the dependencies are updated with the appropriate flavor rather than with the best match flavor to the local machine.

Note: When updating several instances at a time, you can exclude a particular instance from being updated by running **ups touch** on it. See the reference section 22.16 *ups touch* for more information.

10.10 Retrieve an Individual File

The **`upd fetch`** command retrieves a single file or directory maintained in a **UPS** distribution database, and downloads it to the user node, placing it relative to the current working directory. The **`-J`** option is used to specify the individual filename to fetch. If **`-J`** is omitted, the output is a recursive list of directories and files that are available for individual retrieval. Nothing actually is retrieved when **`-J`** is omitted. The full command description and option list is in section 23.6 *upd fetch*. The command syntax with some commonly used options is:

```
% upd fetch [-H <flavor>] [<chainFlag>] [-h <host>] \
  [-J fileName] <product> [<version>]
```

First we issue the **`upd fetch`** command without the **`-J`** option to find out what files are available for the specified product instance (output edited for brevity):

```
% upd fetch -H IRIX+6.2 rbio v9_3d
```

```
Listing of table_dir [/ftp/products/rbio/v9_3d/IRIX+6.2]:
total 3172
drwxrwx---   3 updadmin upd           512 May  7  1999 rbio_v9_3d_IRIX+6.2
-rw-rw-r--   1 updadmin upd          1235 May  7  1999 rbio_v9_3d_IRIX+6.2.table
-rw-rw----   1 updadmin upd       1597440 May  7  1999 rbio_v9_3d_IRIX+6.2.tar
-rw-rw-r--   1 updadmin upd         14848 May  7  1999 rbio_v9_3d_IRIX+6.2.ups.tar

rbio_v9_3d_IRIX+6.2:
total 6
-rw-rw-r--   1 updadmin upd          1540 May  7  1999 README
drwxrwsr-x   5 updadmin upd           512 May  7  1999 ups

rbio_v9_3d_IRIX+6.2/ups:
total 28
-rw-rw-r--   1 updadmin upd           210 May  7  1999 Version
...
rbio_v9_3d_IRIX+6.2/ups/toInfo:
total 0
...
```

Now we use **`upd fetch -J`** to retrieve the `README` file listed. The file will be copied to the current working directory:

```
% upd fetch -J README -H IRIX+6.2 rbio v9_3d
```

```
informational: transferred README
 from fnkits.fnal.gov:/ftp/products/rbio/v9_3d/IRIX+6.2/rbio_v9_3d_IRIX+6.2
 to ./README
```

To verify the successful transfer, we check the current working directory for the new file:

```
% ls -l README
```

```
-rw-rw-r--    1 aheavey  g020        1540 Sep  7 15:57 README
```

As another example, you can retrieve the table files for several flavors of a product. When specifying the flavors on the remote node, be sure to use **`-H`**, not **`-f`**:

```
% upd fetch -H SunOS+5:IRIX+6:Linux+2:OSF1+V4 -J @table_file \
  tex v3_14159
```

This command retrieves the table file(s) for the best match product instances of **tex** v3_14159 for the listed flavor families. Depending on how the product was configured, the same table file may be used for all, or they may be separate files. The file(s) will be copied to the current working directory.

10.11 Check Product Accessibility

The **ups exist** command is used to test whether a **setup** command issued with the same command line elements is likely to succeed. It checks for a properly declared matching instance, and verifies that you have the necessary permissions to create the temporary file used by the **setup** command.[1] This command is rarely used from the command line, and is more useful in scripts where a failed setup could cause the script to abort. The full command description and option list is in section 22.7 *ups exist*. The command syntax with some commonly used options is:

```
% ups exist [-f <flavor>] [<chainFlag>] [-j] <product> \
  [<version>]
```

When issued from the command line, it returns no output if the command succeeds. In the C shell family **ups exist** sets the $status variable to `0` if it was able to create the temporary file, or to `1` for error. In the Bourne shell family, it sets the $? variable similarly. As an example, we can run **ups list** (not shown here) and find that there is a current instance of the product **tex** for the flavor IRIX+6 but not for IRIX+6.2. Running **ups exist** for each flavor, we see that the variables get set accordingly. For the C shell family:

```
% ups exist tex -f IRIX+6; echo $status
    0
% ups exist tex -f IRIX+6.2; echo $status
    1
```

For the Bourne shell family:

```
$ ups exist tex -f IRIX+6; echo $?
    0
$ ups exist tex -f IRIX+6.2; echo $?
    1
```

To run this on a product distribution node, use the corresponding command **upd exist**, documented in section 23.5 *upd exist*.

1. Specifically, it determines whether **setup** can create the temporary file. If so, it creates it, but it does not execute it.

10.12 Troubleshooting

This section provides a few hints if things don't seem to work after declaring/removing/changing a product, or otherwise modifying files in a **UPS** database.

- If the $PATH goes away, restore it by running:

  ```
  % setup setpath
  ```

  and check if the **pathSet** function is used in the table file -- if it is set wrong, this may be the cause.

- To print out diagnostic information about what might be wrong with a product declaration, run **ups verify**:

  ```
  % ups verify -a <product> [<version>]
  ```

- Try setting up just the main product and none of its dependencies. This should help determine which file has the problem, the main one or a dependency. Use **-j** in the **setup** command:

  ```
  % setup -j <product>
  ```

- Print out verbose information using the **-v** option with **setup**:

  ```
  % setup -v <product>
  ```

  To get progressively more information, use multiple **v**'s, e.g., **-vv**, **-vvv** (up to four).

- Check file permissions. Any scripts called by the table file must be both readable and executable. The product executable(s) must of course be executable. The product database files must be readable.

- To examine the temporary file that the **setup** command creates and sources, run the command:

  ```
  % ups setup <product> [<version>]
  ```

  This returns the path of this temporary file, and you can then go look at the file. For example:

  ```
  % ups setup ocs
  /var/tmp/aaaa00273
  ```

- For most **UPS** commands, the **-s** option can be used to simulate the command (i.e., create the temporary file) without executing it. It also returns the path of the temporary file it created, for example:

  ```
  % setup -s -z /products/ups_database/upsII/main xpdf
  INFORMATIONAL: Name of created temp file is /var/tmp/aaaa005Mt
  ```

- If home directories move or if older versions of products have been deleted, you might want to prevent execution of unsetup files prior to a subsequent setup. In this case, don't **unsetup** the product. Just setup the product again using **-k**:

  ```
  % setup -k <product>
  ```

Chapter 11: UPS and UPD Pre-install Issues and General Administration

In this chapter we take a step back with regard to Chapter 10: *Maintaining a UPS Database*, and assume that you have not yet installed **UPS/UPD**, or created a **UPS** database and products area. We guide you through the administrative decisions and tasks that are involved in preparing to implement **UPS/UPD**. Towards the end of the chapter there is also some information regarding general administrative tasks. For machines running AFS or NFS, also see Chapter 12: *Providing Access to AFS Products*.

11.1 Choosing Installer Accounts

Here we assume that you're planning to implement **UPS/UPD** on your local system and maintain a **UPS** database and products area there[1]. In this section we discuss options regarding installer accounts. Choosing installer accounts wisely is important because the account used by the installer determines a product's ownership.

11.1.1 Single Installer Account

For a given system, if the number of people who install products from a distribution node and manage the local **UPS** database is small and relatively static, then, from a system management point of view, having a single account used for these purposes is often simplest. We suggest you create a standard account called *products* to be used for all product installations, then no special permissions are needed for other accounts. This single account method prevents problems for any product maintainer who has to remove or change a product installed by a different person. As far as the system is concerned, the installer/maintainer is always *products*.

11.1.2 Multiple Installer Accounts

Our experience has shown that in many situations a single installer account is not adequate. It leads to confusion on systems where several people install products, especially if they don't establish and follow a procedure for communicating with each other regarding product maintenance. If they all use the *products* account, it is much more difficult to track which person performed a particular operation.

1. We specifically don't say "planning to install **UPS/UPD**" because in some configurations (notably AFS machines), you may not need to install the products locally.

As systems grow in size, and more and more users become product installers, we have found that it is better for them to use their normal login accounts when installing products. Since installers need the correct group id to write to the products area (often this group is the *products* gid), system managers can simply add them to the *products* group.

This strategy does warrant more caution in two areas: file system semantics and **UPD** configuration, as described in sections 11.2 *Setting gids for Multiple Installer Accounts* and 11.3 *File Ownership, Permissions and Access Restrictions*.

11.1.3 Separate Installer Accounts for Different Product Categories

You might want the ownership of the product home area to be different based upon whether the product is being downloaded from a distribution node, or was developed on the local machine.

For example, say you have one person who installs all the products needed on your system from the distribution node, and several people developing **UPS** products locally on your system. In this case, you may find it simplest to use the single installer *products* account for the downloaded products, and the developers' own accounts for the locally-developed products.

11.2 Setting gids for Multiple Installer Accounts

When you allow multiple accounts to install products as described in section 11.1.2 *Multiple Installer Accounts*, you use group ids (gids) to control access to the product areas. Group ids get set on files and directories differently depending on whether you use System V or Berkeley semantics (the choice for most current Unix systems). With System V semantics, new directories and files have the same gid as the account which created them. With Berkeley semantics, new directories inherit the gid of the parent directory.

If you plan to allow product installs from multiple accounts, we strongly recommend using Berkeley semantics. On newer systems, you select Berkeley by setting the set-group-id bit on the directory (i.e., **`chmod g+s <directory>`**). On older systems this may require special options on the filesystem mount.

UPD unwinds tar files with the permissions given when they were created. Most of the product files do *not* have group-write enabled. You might find it useful to make the products you install group-writable so that the same set of accounts that installs products can also delete them. You will be able to set this in your local **UPD** configuration file (`updconfig`, described in Chapter 31: *The UPD Configuration File*) in a `postdeclare` action, e.g.,:

```
ACTION = POSTDECLARE
    Execute("chmod -R g+w ${UNWIND_PROD_DIR}", NO_UPS_ENV)
```

Whether products are downloaded from a server or developed locally, the group ownership is still an issue. For a locally developed product, sometimes many accounts need to be able to delete or modify it, and the group access needs to be set accordingly. There is no automatic way in **UPS/UPD** to enforce this, however.

11.3 File Ownership, Permissions and Access Restrictions

11.3.1 Product Files

Product file ownership is determined by the installer account, as discussed in section 11.1 *Choosing Installer Accounts*. To determine adequate file permissions for **UPS** products, it is important to consider your user community. Clearly users will need read access to the products they are allowed to use, but do you need to restrict access to any subsets of products? Do all users share the same group ids? Who needs to install/delete products? Who will be declaring products to the **UPS** database? These are the kind of questions you should keep in mind when setting up file permissions.

Some products, e.g., licensed products, should not be accessible to all users. A simple way to restrict user access is to configure a special **UPS** database for your restricted products and make that database visible only to the appropriate subset of users. Sometimes you may need to protect the files themselves, as well. When extra security is required, we recommend that you create a special gid for the restricted products and that you turn off world access. You will be able to configure **UPD** to set this group id on those products; this example shows how it can be done in `updconfig` (file described in Chapter 31: *The UPD Configuration File*):

```
group:
    product = some_licensed_product
common:
    UPS_PROD_DIR = ...
    UNWIND_PROD_DIR = ...
    ...
    action = postdeclare
        Execute("chgrp -R <special_group> ${UPS_PROD_DIR}", NO_UPS_ENV)
        Execute("chmod -R o-r ${UPS_PROD_DIR}", NO_UPS_ENV)
end:
```

A stanza like this in the `updconfig` file should be otherwise identical to the existing default stanza that would have handled this product; only the `postdeclare` action should be added. Within the `updconfig` file, the default stanza should come *after* any specialized stanzas, since **UPD** uses the first match it encounters.

11.3.2 Database Files

The **UPS** database files and pointers therein (e.g., to man page areas) have their own set of file permissions that are generally more open than the product files themselves. In almost all cases, the **UPS** database files should be group-writable. Set your `umask` to `002` before running **`upd install`** or **`ups declare`** on products to ensure this. The files should be owned by an account with a gid that is shared by the installation account(s) and by any developers creating **UPS** products locally.

11.4 Product File Location and Organization

11.4.1 Considerations

UPS/UPD imposes no restrictions on where product files can reside. Product files can reside on any file system on any disk. Different instances of the same product can reside in different file systems. **UPD** allows you configure where you want specific products unwound. The more generic your rules, the simpler your **UPD** configuration file (`updconfig`, described in Chapter 31: *The UPD Configuration File*).

As a system manager, you should check how much space is available in each partition. If you have many machines sharing disks, determine how they are shared. Do they have the same mount points on all machines? Is the `/usr/local` area shared across any machines? Are you cross-mounting to different OS flavors? How should products be organized in the file system? What do you want to keep local to a particular machine, local to all nodes of particular flavor, or shared cluster-wide?

Take into account these issues when deciding where to put product files, as well as databases and other files. Your system configuration affects how many copies of a given product instance may be required on a given group of systems. It also determines on how many separate machines a product instance may require that special actions be performed (e.g., configuring or declaring it current).

11.4.2 Single Flavor or Single Node Systems

For simple systems, e.g., single flavor or single node, the product files typically live under:

```
/path/to/<product_name>/<product_version>
```

However, this is not always adequate. An example is if you have more than one instance of the same version, but with different qualifiers. In cases like this, the following is a better model:

```
/path/to/<product_name>/<version><qualifiers>
```

To implement this, all you need is one stanza in your `updconfig` file which lays out all the products in this way.

However, experience has shown us that it's not always wise to assume that your system will forever have only one flavor; system upgrades are not so predictable. See the next section, 11.4.3, for more ideas.

11.4.3 Multi-Flavor and/or Multi-Node Systems

Two separate directory structure conventions have evolved for products installed on multi-flavor systems. Each has its own advantages and disadvantages.

1) `/path/to/<product_name>/<flavor>/<version><qualifiers>`

   In this configuration, all versions of a product are unwound on the same file system. This makes it easy to see what software is available using simple UNIX **`ls`** commands.

2) `/path/to/<base_flavor>/<product_name>/<version><qualifiers><flavor>`

   This second case supports a separate file system for each operating system. If one disk is currently not available, work can still continue on other machines.

Note that in both cases, `/path/to/` can be anything, and need not be the same for each product (but in general it's easiest to maintain all products in the same area). If more than one file system is used, **UPD** needs to be configured to know which products are installed in which file system. If the products need to be seen across multiple machines, it is important that all the file systems be visible under the same directory structure on all machines.

Whichever structure you choose, when you have multiple flavors (or possible future multiple flavors) you may find it useful to create a flexible configuration that allows **UPS/UPD** to pick a different product directory based on flavor.

The following is a sample `updconfig` file showing how to do this. The flavors of products we expect to install, in this case NULL and IRIX+6 products, get put in with nice, short pathnames (see `UPS_PROD_DIR` in the file), while everything else gets a longer pathname that explicitly includes the flavor.

```
file=updconfig
group:
 flavor=NULL
 flavor=IRIX+6
common:
 UPS_THIS_DB = "/fnal/ups/db"
 UPS_PROD_DIR = "${UPS_PROD_NAME)/${UPS_PROD_VERSION}${UPS_DASH_QUALIFIERS}"
 UNWIND_PROD_DIR = "${PROD_DIR_PREFIX}/${UPS_PROD_DIR}"
 UPS_UPS_DIR = "ups"
 UNWIND_UPS_DIR = "${UNWIND_PROD_DIR}/${UPS_UPS_DIR}"
 UNWIND_TABLE_DIR = "${UPS_THIS_DB}/${UPS_PROD_NAME}"
 UPS_TABLE_FILE = "${UPS_PROD_VERSION}.table"
end:
group:
 flavor=ANY
common:
 UPS_THIS_DB = "/fnal/ups/db"

UPS_PROD_DIR="${UPS_PROD_NAME)/${UPS_PROD_VERSION}${UPS_DASH_FLAVOR}${UPS_DASH_QUALIF
IERS}"
 UNWIND_PROD_DIR = "${PROD_DIR_PREFIX}/${UPS_PROD_DIR}"
 UPS_UPS_DIR = "ups"
 UNWIND_UPS_DIR = "${UNWIND_PROD_DIR}/${UPS_UPS_DIR}"
 UNWIND_TABLE_DIR = "${UPS_THIS_DB}/${UPS_PROD_NAME}"
 UPS_TABLE_FILE = "${UPS_PROD_VERSION}.table"
end:
```

11.5 Database File Location and Organization

11.5.1 Choosing Single or Multiple UPS Databases

The **UPS** database files are not large and do not require much disk space. They can reside on any file system; either with the product files, or not. You can choose to have one **UPS** database for all products installed on a mixed flavor cluster, you can have a separate **UPS** database for each flavor type, or you can choose another configuration. Typically, a single **UPS** database is used for all flavors. Multiple databases are more often used to house sets of user-specific products (e.g., CDF off-line products) rather than to distinguish between operating system flavors.

11.5.2 UPS Database File Pointers

A **UPS** database contains pointers to directories, the most important of which are the ones that contain the man page files and the **UPS** environment initialization files (`setups.[c]sh`; discussed in section 1.7.1 *Initializing the UPS Environment*). These directories can be on different file systems from the database itself. In order to best determine the locations and permissions for these directories, system administrators need to understand how they are used.

When a product is declared current, its man pages, if any, are (optionally) copied to a system-wide **UPS** product man page directory for which the location is set in the database configuration file, `dbconfig`. This man page directory should be writable by anyone with the authorization to declare products current. Historically, this directory has been maintained separately from normal system man pages, just to avoid any confusion or overlap.

Often this **UPS** man page area is shared between OS flavors. This is an easy solution, but it can lead to confusion on mixed OS clusters. If you have different versions of a product declared current for each flavor, the man pages will likely get out of sync. For example, say you have an installed IRIX “current” chain for a product, and you later declare a SunOS instance “current”. If the man page area is shared, this new man page overwrites the older one.

The `setups.[c]sh` files make the **UPS** environment available by defining the **UPS** database(s) and setting up **UPS** itself. These files are invoked by each individual user’s login scripts, and their location is configured in the **UPS** database configuration file `dbconfig`. In the past, these files have been kept in `/usr/local/etc`; however, this has been a problem on machines where the person installing **UPS** does not have *root* access. A more common practice now is to put them in a directory parallel to the main **UPS** database itself, e.g., `$PRODUCTS/../etc`.

11.6 Installing UPS for Use Without a Database

UPS can be installed on a machine to manage products without a **UPS** database, as mentioned in section 1.5 *Using UPS Without a Database*. This flexibility is provided primarily for off-site users who for one reason or another do not want to maintain a **UPS** database on their local system.

Before making the decision to do without a **UPS** database, be aware that **UPS** allows much more flexibility now. **UPS** is no longer tied to products such as **futil** and **systools**, and the database can be maintained anywhere on your system.

To install **UPS** in this way:

1) Create a products area (not strictly necessary, but this keeps things more organized).

2) Download **UPS** into the products area using **FTP**, as described in Chapter 7: *Installing Products using FTP*.

3) Initialize your **UPS** environment as described in section 1.7.1 *Initializing the UPS Environment*.

A locally installed product instance would have no version or chain files, of course, but it would need a table file (very few products come without one). If there are any functions in the table file for setting up product dependencies (e.g., the function `setupRequired` or `setupOptional`), you'll need to check each of these functions to make sure it includes a table file specification. This is necessary in order to continue to bypass the need for a database.

Make sure your users are made aware that:

- **UPS/UPD** functionality requiring or operating on a **UPS** database is not supported when **UPS** is implemented without a database (e.g., **`setup`** should work, but **`ups declare`** won't).
- Any **UPS/UPD** command must include all of the information that normally would have been read from a database. In particular, all commands require the **`-m`** option for table file name (and usually **`-M`** for the table file directory).

11.7 CYGWIN (Windows NT) Issues

A number of **CYGWIN**-specific problems have been encountered. We'll highlight the most frequent ones here.

11.7.1 Using Correct Perl Version

You must run a version of **perl** that is built against **CYGWIN** itself, not one arbitrarily obtained off the net. You can get a working **perl** for **CYGWIN** from `KITS`.

11.7.2 Mounting the CYGWIN bin Directory

The **CYGWIN** `bin` directory should be mounted in `/usr/bin` and a symbolic link must be made from `/usr/bin` to `/bin`.

11.7.3 Setting Environment Variables

For **UPD** to work properly, make sure that:

- $TMPDIR is set to a directory that really exists and that can be used for holding temporary files
- $USER is set to your userid

11.8 General Administration Issues

11.8.1 Upgrading an Older System

Prior to **UPS/UPD** v4, the Fermi User Environment (FUE) included a suite of utilities and binaries which were copied into `/usr/local`. The required FUE utilities, with a brief description of each, were:

systools system administration utilities, located under `/usr/local/systools`; the utilities included the Fermi login scripts and ups initialization files, as well as the **`cmd`** function (which allows non-privileged users to do specific privileged things) and various "**`cmd <utilities>`**", e.g., **`adduser`**, **`renice`**, etc.

funkern programs called from the Fermi login scripts, copied into `/usr/local/bin`

fulib C-callable library of the **funkern** utilities

futil a random assortment of odds and ends that were considered to be Generally Useful Utilities, copied into `/usr/local/bin`

FUE has recently been redesigned utilizing the new features of **UPS** v4. The redesign has been very successful, and is in use on all FUE-compliant systems installed since the autumn of 1999. To upgrade an older system and clean up the vestiges of the old FUE, you must first upgrade to the new FUE which includes **UPS/UPD** v4_0 or higher (v4_5_1 as of May 2000), and **systools** v6_0 or higher (and all of its dependencies). You must also convert existing accounts' login scripts to the new FUE syntax. Two technical notes are available to guide you through this process. See:

TN0088 *Files which you may be able to remove from /usr/local* (`http://www.fnal.gov/docs/TN/tn0088.html`)

TN0089 *FUE Login Methodology* (`http://www.fnal.gov/docs/TN/tn0089.html`).

11.8.2 Adding a New Database and/or Products Area

There are different reasons for adding a new products area. For example, you may run out of room in one area and need another, or you may want different categories of products stored in different areas and accessible to different groups of people. For development and/or testing purposes, it is often convenient to install products in a separate products area and declare them in a separate, associated **UPS** database.

Checklist

Here is a checklist of the tasks involved (assuming **UPS/UPD** is already installed and working on your system):

- Create a directory for the **UPS** database (e.g., `/path/to/db`).
- Create a products area
- Create a **UPS** configuration file (`/path/to/db/.upsfiles/dbconfig`).
- Create a **UPD** configuration file (`path/to/db/.updfiles/updconfig`).
- Create an empty dummy directory under the database (e.g., `/path/to/db/xxx`).[1]
- Prepend your database path to $PRODUCTS; colon separated (e.g., `setenv $PRODUCTS "/path/to/db:${PRODUCTS}"`).
- If you are at **UPS** version v4_4a or earlier, include the file `updusr.pm` under `/path/to/db/.updfiles`. Insert as the file contents the following single line: `require 'default_updusr.pm';`

Example

Here we provide an example of creating a new product area (we'll use `/fnal/ups/prd`) complete with a new database (`/fnal/ups/db`). We've created a shell script called `newupsarea`. To run it, we must supply the directory which houses the database and products area (`/fnal/ups`) as an argument. First, let's look at this script:

```
#!/bin/sh

db=$1/db
pdp=$1/prd
mkdir -p $db/.upsfiles
mkdir -p $db/.updfiles
mkdir -p $pdp
cp $UPD_DIR/ups/updconfig.template $db/.updfiles/updconfig
cp $UPD_DIR/ups/updusr.pm.template $db/.updfiles/updusr.pm
cp $UPS_DIR/ups/dbconfig.template  $db/.upsfiles/dbconfig
perl -pi.orig -e 's{/fnal/ups}{'$1';' $db/.upsfiles/dbconfig
# work around empty db bug...
mkdir $db/xxx
```

To run this, issue the command:

```
% newupsarea /fnal/ups
```

This creates the database (`/fnal/ups/db`), its subdirectories, the PROD_DIR_PREFIX directory (`/fnal/ups/prd`), copies the **UPS** and **UPD** configuration files from templates, and changes the references to `/fnal/ups` in the copy of the template `dbconfig` file. Finally, it makes a directory (`xxx`) under the database[2].

1. Given a database with the subdirectories `.upsfiles` and `.updfiles` and nothing else, the command **`ups list -K PROD_DIR_PREFIX`** fails to list that database and its prefix. In turn, **UPD** fails to find PROD_DIR_PREFIX for that database, and so on. Simply adding an empty subdirectory solves the problem.

11.8.3 Collecting Statistics on Product Usage

UPS supports recording of the following statistics on product usage and **UPS** database access:

- Userid of person executing **UPS/UPD** command
- Date and time
- Which command was executed (including options and arguments)
- Which product instance was selected by command

Collection of statistics is controlled by an entry in one or more database files. See the reference section 27.6.3 *STATISTICS* for a description of how to implement this.

2. Given a database with the subdirectories `.upsfiles` and `.updfiles` and nothing else, the command **`ups list -K PROD_DIR_PREFIX`** fails to list that database and its prefix. In turn, **UPD** fails to find PROD_DIR_PREFIX for that database, and so on. Simply adding an empty subdirectory solves the problem.

Chapter 12: Providing Access to AFS Products

This chapter describes how to provide access on your local machine to **UPS** products installed in AFS space.

12.1 Overview

Much of the information in this chapter is adapted from document number TN0091, *Configuring a Local UPS Database (While Still Using the Centrally Supported AFS database)*, found on the Web at `http://www.fnal.gov/docs/TN/tn0091.html`.

To minimize duplicate effort in supporting software, a centrally-supported **UPS** database in AFS space is maintained by the product developers. Systems running AFS are encouraged to use the AFS **UPS** database for the majority of their software needs. However, there are cases where a local database is needed in addition to the AFS database (for locally maintained or developed software, different version requirements, and so on).

A system in AFS space does not need to run the bootstrap procedure documented in Chapter 13: *Bootstrapping CoreFUE*. **UPS**, **UPD**, and **perl** (these products together are referred to as **CoreFUE**) are already available to you via the **UPS** database in AFS space.

You can configure your system for a number of different options regarding AFS product availability:

- a local **UPS** database but no local **CoreFUE** installation, providing access to local and AFS products
- a local **UPS** database and local **CoreFUE** installation, also providing access to local and AFS products
- no local **UPS** database (nor local **CoreFUE** installation), providing access to AFS products only

Whether you want to maintain a local database or not, if you want access to the **UPS** products in AFS space, you need to update your `/usr/local/bin` area as shown in section 12.5 *Updating /usr/local/bin to Access AFS Products*.

For those of you who choose to maintain a local database, we recommend that you *not* install **CoreFUE** locally unless it is absolutely necessary. In most cases, the disadvantages (extra product maintenance responsibilities and a more complicated configuration) considerably outweigh the benefits (access to products when AFS is down and more flexibility in file naming conventions).

Note that the concepts discussed here are equally applicable to local **UPS** databases on machines in an NIS cluster with its own common NFS-mounted database. You must make suitable modification to the particular details, e.g., wherever you see `/afs/fnal.gov/ups`, replace it with the appropriate path to the NFS-mounted area, e.g., `/fnal/ups`.

12.2 Configuring a Local Database to Work With AFS

Here we describe how to configure your system to provide access both to locally installed products, declared in a local **UPS** database, and to products in the AFS-space **UPS** database.[1] In this and following subsections, $PARENT_DIR refers to the local directory under which all the **UPS** database files and product files reside.

Note ahead of time that there are several local configurations preset in AFS space (i.e., the AFS `$SETUP_DIR/upsdb_list` file recognizes these locations). We recommend that you choose one of them. In fact, if the local database and products area are put in a non-preset location, then this scheme becomes much harder to implement without a local copy of **UPS/UPD**; see section 12.4 *Additional Steps for Unfamiliar Naming Conventions*. In general, the recommended directory under which all the **UPS** database files and product files reside ($PARENT_DIR) is:

`/fnal/ups`	the standard naming convention provided by several bootstrap configurations for product server nodes

Other preset configurations:

`/local/ups`	standard provided by the Fermi RedHat Linux bootstrap for satellite nodes
`/usr/products`	another popular naming convention
`/usr/products/CMSUN1`	standard for CMS local databases

12.2.1 Steps to Create and Configure the Database

1) Create the top-level directory ($PARENT_DIR). Make sure that your *products* account (or whichever account should own the product files) can read and write to this directory.

2) Log in as *products*, **`cd`** to $PARENT_DIR and create the following directories:

`% mkdir db`	contains the database
`% mkdir db/.upsfiles`	local **UPS** configuration file goes here
`% mkdir db/.updfiles`	local **UPD** configuration file goes here
`% mkdir prd`	local product file hierarchy begins here
`% mkdir man`	local man pages go here
`% mkdir catman`	local catman pages go here

1. The astute reader will notice that there are an infinite number of alternatives; the steps shown are, however, sufficient for most purposes.

3) Create the **UPS** database configuration file for your local database:

```
% setup ups
```
using the copy of **UPS** in AFS space!

```
% cd $PARENT_DIR/db/.upsfiles
```
change to the location of your local **UPS** configuration

```
% cp $UPS_DIR/ups/dbconfig.template ./dbconfig
```
copy the template `dbconfig` file from the **UPS** in AFS space to your area

Edit your local `dbconfig` file and replace `/fnal/ups` with your $PARENT_DIR, if different from `/fnal/ups`.

4) Create the **UPD** configuration file for your local database:

```
% setup upd
```
using the copy of **UPD** in AFS space!

```
% cd $PARENT_DIR/db/.updfiles
```
change to the location of your local **UPD** configuration

```
% cp $UPD_DIR/ups/updconfig.template ./updconfig
```
copy the template `updconfig` file from the **UPD** in AFS space to your area

In most cases the default `updconfig` file should be perfectly adequate.

5) Create the FUE initialization files for your system. These are the files that will be called when users log in (or when other processes start) in order to initialize the FUE environment.

If your configuration follows a well-known naming convention (`/fnal/ups`, `/local/ups`, `/usr/products` or `/usr/products/CMSUN1`) you can take advantage of the configuration already maintained in AFS space by creating symbolic links that you never need to modify again (here we assume that $PARENT_DIR is set to `/fnal/ups`):

```
% cd $PARENT_DIR
```
change to your $PARENT_DIR

```
% ln -s /afs/fnal.gov/ups/etc ./etc
```
this makes your $SETUPS_DIR a link to AFS

If you are creating the “courtesy links”, you should log in as *root* and run the following commands:

```
% cd /usr/local/etc
% ln -s /afs/fnal.gov/ups/etc/setups.csh ./setups.csh
% ln -s /afs/fnal.gov/ups/etc/setups.sh  ./setups.sh
```

Your local database is now configured.

If your configuration does not conform to a well-known convention, please refer to section 12.4 *Additional Steps for Unfamiliar Naming Conventions.*

12.2.2 Post-Configuration: Reinitialize FUE Environment

To use your configured database, reinitialize the FUE environment for your process by running:

For the C shell family:
```
% unsetenv PRODUCTS
% source $PARENT_DIR/setups.csh
```

For the Bourne shell family:
```
$ unset PRODUCTS
$ . $PARENT_DIR/setups.sh
```

Your $PRODUCTS environment variable should now include both databases (with the local database coming before the AFS database).

12.2.3 A Note about Product Installation for this Configuration

Users of this type of configuration typically install products using **UPD** (see Chapter 5: *Installing Products Using UPD*). Make sure that the product installers on your system know that to install a product into the local database, they must use the AFS installation of **UPS/UPD**, but **UPD** must use your local configuration files. This is very important! Assuming that $PRODUCTS lists your database first, and the product in question doesn't exist in the AFS database, you can run **`upd install`** without any database option and your product will go into the local database.[1] Otherwise, include the **`-z`** option in the **`upd install`** command, e.g.,:

```
% upd install -z /fnal/ups/db[:other-dbs] ...
```

12.3 Installing a Local Copy of CoreFUE

Recall that we discourage installing and maintaining **CoreFUE** locally when the machine is running AFS. The pros and cons are spelled out in document number TN0091, *Configuring a Local UPS Database (While Still Using the Centrally Supported AFS database)*, found on the Web at `http://www.fnal.gov/docs/TN/tn0091.html`.

To install **CoreFUE** locally, first create and configure your local **UPS** database as outlined in section 12.2 *Configuring a Local Database to Work With AFS*. Use the AFS installation of **UPD** to install **UPS**, **UPD** and **perl** into the local database (yes, **UPD** can install itself elsewhere). Then, in your `$SETUPS_DIR/upsdb_list` file ($SETUPS_DIR is set in the `dbconfig` file), make sure that you include/activate the line:

```
/afs/fnal.gov/ups/db
```

1. This also assumes the local `updconfig` file says to install in the local database.

A Note about Product Installation for this Configuration

Whenever you use **UPD**, set up the instance in the local database to ensure that it uses your local `updconfig` file by default. If you set up the AFS installation of **UPD**, you can use **`upd install -z /path/to/yourdb[:other-dbs]`** to make it use the local configuration.

12.4 Additional Steps for Unfamiliar Naming Conventions

If your **UPS** database configuration does not conform to one of the well-known conventions in AFS space, you will need a way of making sure that your local **UPS** database is included in $PRODUCTS. There are three ways to accomplish this:

1) Lobby to be added to the list of well-known conventions. Send mail to *ups@fnal.gov* stating the name of the local **UPS** database, and a good reason why it should be mentioned in the lab-wide AFS `upsdb_list` (list of known local databases).

2) Use $UPS_EXTRA_DIR. Make sure that everybody who needs access to your local **UPS** database modifies all of their login scripts (and other scripts) to set the $UPS_EXTRA_DIR environmental variable to your database before they source the `setups.[c]sh` script. This is a viable alternative if there is only a small community of people who need this database (e.g., a small group of developers on your local system).

   For example:

   ```
   # Cshell example of $UPS_EXTRA_DIR
   #
   setenv UPS_EXTRA_DIR /our/unfamiliar/local/db
   source /afs/fnal.gov/ups/etc/setups.csh
   ```

   Any directories in $UPS_EXTRA_DIR will be prepended to the database directories listed in the `upsdb_list` file the first time you source the `setups.[c]sh` script. (Of course, you can always prepend the appropriate database to your $PRODUCTS manually at any time).

3) Install the components of **coreFUE** locally (**UPS**, **UPD** and **perl**). Maintain your own version of the `setups.[c]sh` scripts by installing a local copy of the **coreFUE** product into your database. Setup the AFS space **UPD** product:

   `% setup upd`

   Use this to install coreFUE into your local database and chain it to current:

   `% upd install coreFUE -z $PARENT_DIR/db -G -c`

Then make sure that your copy of the `$SETUPS_DIR/upsdb_list` contains all of the directories that you wish to include in $PRODUCTS (including the AFS **UPS** database).

If you are creating the “courtesy links”, you should log in as *root* and issue the commands ($PARENT_DIR is the common parent directory for the local database and products area):

`% cd /usr/local/etc`

`% ln -s $PARENT_DIR/etc/setups.csh ./setups.csh`

```
% ln -s $PARENT_DIR/etc/setups.sh  ./setups.sh
```

Remember, you will need to keep these local copies of **UPS**, **UPD**, **perl** up to date!

12.5 Updating /usr/local/bin to Access AFS Products

Whether you configure a local **UPS** database or not, if your machine runs AFS and you want access to any AFS-space **UPS** products, you need to update certain files (or links to files) in your local `/usr/local/bin`. These required links and/or files are associated with programming shell and login shell products in the AFS-space **UPS** database. Currently the list of products that require files or links to files in `/usr/local/bin` are: **perl**, **tcsh**, **bash**, **python** (if you need it), and **systools**[1] (see section 8.1 *Installing Products that Require Special Privileges* for more information). We are trying to minimize the number of products which write into `/usr/local`, and we hope the process of updating this area won't be necessary in the future.

Here's how to update your local `/usr/local/bin`:

1) Mount `/afs` by running `mount /afs`.

2) Set the variable $PRODUCTS to your local database.

3) You need your local node to point to the **UPS** in AFS space. If you've configured a local database, you've probably already done this step. If not, login as *root* and issue the following commands to set your `/usr/local/etc` courtesy links to point to `/afs/fnal/ups/etc/setups.[c]sh`:

```
% cd /usr/local/etc

% ln -s /afs/fnal.gov/ups/etc/setups.csh .

% ln -s /afs/fnal.gov/ups/etc/setups.sh  .
```

4) Then, still logged on as *root*, update your `/usr/local`, by running the following commands (from any directory):

```
% source /usr/local/etc/setups.csh
```

(or `$ . /usr/local/etc/setups.sh` for Bourne shell)

```
% ups installasroot perl

% ups installasroot bash

% ups installasroot tcsh

% ups installasroot python
```

1. As of this writing, **systools** is not really part of this list, but we expect it to be added.

Chapter 13: Bootstrapping CoreFUE

CoreFUE is a bundled product which includes **UPS/UPD** and **perl**. It refers to the core components of the Fermi UNIX Environment (FUE).[1] When we discuss installing **UPS/UPD**, we're generally talking about **coreFUE** since **perl** is a required component. Here we describe how to use automated scripts to bootstrap **coreFUE**, that is, to install **coreFUE** on a machine on which no prior versions of these products are installed. Several project-specific configurations of **UPS**/**UPD** are available.

The v2_0 version of the bootstrap for **UPS** has been significantly streamlined and is less error-prone than the preceding release. Automated installs are available for UNIX and NT (with **CYGWIN**). The UNIX install requires about 50M, and the NT about 70M. You can choose a pre-defined configuration and use it as is or edit it, or you can define your own configuration. Alternatively, it is possible to run a manual installation from a bootstrap tar file (this option not documented here; see *Bootstrap CoreFUE Installation Summary* at `ftp://ftp.fnal.gov/products/bootstrap/v2_0/manual_install.html`).

If you plan to run **UPS** without a database (as discussed in section 11.6 *Installing UPS for Use Without a Database*), don't use the bootstrap procedure. Just download **UPS** using **FTP**, as described in Chapter 7: *Installing Products using FTP*.

13.1 Downloading the Bootstrap and Configuration Files

The bootstrap script for UNIX is called `stage1.sh`. For NT it is called `stage1.bat`. They are both available for download from `ftp://ftp.fnal.gov/products/bootstrap/v2_0/`. Besides that, the only other file you need to download is a configuration file. There are several configuration files from which to choose, as described below.

13.1.1 Predefined Configurations for UNIX

These configurations are intended mainly for on-site users with administrative privileges on their systems. Choose one of the following customized configuration files found under `ftp://ftp.fnal.gov/products/bootstrap/v2_0/configs/`:

1. Another bundled product you should know about is **FullFUE**. It consists of **coreFUE** plus **systools**, **sectools**, **futil**, **login_shells** and some "courtesy links". FUE is described in the document DR0009, available at `http://www.fnal.gov/docs/Recommendations/dr0009.html`.

`local`	puts a products area under `/local/ups`, creates a *products* account if needed, and installs **fullFUE** if it doesn't exist in other standard products areas (e.g., `/afs/fnal/ups`, `/fnal/ups`)
`generic`	puts a products area under `/fnal/ups`, creates a *products* account if needed, and installs **fullFUE** if it doesn't exist in other standard products areas (e.g., `/afs/fnal/ups`, `/fnal/ups`)
`D0`	sets up the standard three D0 RunII development **UPS** areas (`/usr/products`, `/d0usr/products`, and `/d0dist/dist`), creates a *products* account if needed, installs other products to facilitate D0 development (**cvs**, **d0cvs**, **python**) and installs **fullFUE**
`test`	sets up a minimal **UPS** area under `/tmp/ups`

13.1.2 User-defined Configuration for UNIX

To create your own configuration file you'll need the `configurator` script, available from `ftp://ftp.fnal.gov/products/bootstrap/v2_0/configurator`. Once it's downloaded, run the script by issuing the command **`sh configurator`** and answer the questions. This generates a user-customized configuration file called `config.custom`. Here is a sample session:

```
% sh configurator
```

```
Should we put symlinks and login shells in /usr/local[Yn]? n
Do you want to use the existing AFS products area[Yn]? n
Do you want to use any other existing products areas[yN]? n
Do you want to create a database local to this system[Yn]? y
What is the path to the database[/fnal/ups/db]? /scratch/mengel/products/db
Of the following databases:
    1/scratch/mengel/products/db
From which one do you want to get coreFUE (ups, upd, etc.)[1-1]? 1
Should we install the full FUE environment in the local database[yN]? y
Warning -- installing fullFUE without writing in /usr/local may not work
Can we write in /usr/local after all[Yn]? n
Are you sure we should do fullFUE[yN]? n
Writing config.custom
```

13.1.3 Predefined Configurations for NT

These configurations are mainly targeted at D0 and SVX Run II developers. Download one of the following customized configuration files from `ftp://ftp.fnal.gov/products/bootstrap/v2_0/configs/`:

`SVX`	puts a `products` area under `C:\products`
`D0cygC`	puts the three D0 Run II `products` areas under `C:\D0RunII`
`D0cygD`	puts the three D0 Run II `products` areas under `D:\D0RunII`
`D0cygE`	puts the three D0 Run II `products` areas under `E:\D0RunII`

13.2 Customizing a Bootstrap Configuration

If you plan to use one of the available configuration files as is, skip down to section 13.3 *Running the Bootstrap Procedure*.

The bootstrap process includes only steps listed in the specified configuration file[1]. Customizing the bootstrap process therefore only involves editing the configuration file you've chosen. Notice that the configuration file contains several types of instructions. The statements are grouped by type and executed in the order shown below. We recommend that you keep them in this order as you edit your configuration file.

13.2.1 Bootstrap Configuration File Statement Definitions

`# ...`
: Comments

`set_variable <variable>="<string>"`
: Sets variables for the script to use. There are two required variables in the configuration file:

  `bootbase` URL of bootstrap files

  `upsdb_list` **UPS** database list; separate database paths with a space (used to create `$SETUPS_DIR/upsdb_list`). To include the AFS **UPS** database, include `/afs/fnal.gov/ups/db`.

  You can also define your own variables here and use them in later statements.

`check_space </path/to/directory> <size>`
: Verifies that blocks of the specified size are free in `/path/to/directory`.

`download_file <URL> [<local>]`
: Pulls down a file for the script to use. It can be any URL, any protocol.

`pre_install_command "<shell_command>"`
: Runs specified shell command after any **`download_file`** lines and before any **`create_user`** lines

`create_user <name> <uid> <gid> <home>`
: Adds a userid entry to `/etc/passwd`. (This has no effect on NT systems.)

`create_db <path> <dbconfig_file> <updconfig_file> <owner>`
: Creates a **UPS** database. If either of the files is specified as "**`-`**", the command uses the corresponding template file within the downloaded **UPS** product. If the owner is specified as "**`-`**", it uses the current userid.

`install_coreFUE <database>`
: Installs the **coreFUE** product in the specified database, with the owner *products* if possible (it checks the `/etc/password` file for a line starting with `products:`).

---

1. It also puts a temporary **UPS** products area in $TMPDIR, which it cleans out when its done. $TMPDIR defaults to `/var/tmp/bootups`.

`install_as <user> <product> <upd_install_options>`
Runs a **`upd install -G -c <upd_install_options>`** of the specified product, as **`<user>`** if possible (again, it checks the `/etc/password` file for a line starting with `<user>:`).

`do_ups <action>`
Runs an action from the table file of the **UPS** product via the command **`ups <action>`**.

`make_courtesy_links`
Makes links in `/usr/local/etc` to `setups.sh`, etc.

`post_install_command "<shell_command>"`
Runs the specified command, after all the preceding steps are complete.

13.2.2 Sample Customization

In this example, we assume that the bootstrap configuration file `D0` has already been downloaded. We want to edit it such that it prevents the bootstrap from downloading the third listed dbconfig file , `dbconfig3.D0`.

First we create a replacement dbconfig file (we'll call it `/tmp/dbconfig`), and then change the configuration file to refer to it. Let's look at the (abbreviated) file contents prior to the change (affected lines in **bold**):

```
...
#---------------------------------
# files to download
#             remote                                    local
#             -----                                     -----
download_file $bootbase/downloads/updconfig.D0          updconfig
download_file $bootbase/downloads/dbconfig1.D0
download_file $bootbase/downloads/dbconfig1.D0
download_file $bootbase/downloads/dbconfig3.D0
...
#-----------------------------------
# databases
#         database              dbconfig        updconfig       owner
#         --------              --------        ---------       -----
create_db /usr/products/upsdb   dbconfig1.D0    updconfig       products
create_db /d0dist/dist/upsdb    dbconfig2.D0    updconfig       products
create_db /d0usr/products/upsdb dbconfig3.D0 updconfig   products
```

To make the change, we comment out the last **`download_file`** statement and change the name of the dbconfig file in the third **`create_db`** statement:

```
#---------------------------------
# files to download
#             remote                                    local
#             -----                                     -----
download_file $bootbase/downloads/updconfig.D0          updconfig
download_file $bootbase/downloads/dbconfig1.D0
download_file $bootbase/downloads/dbconfig1.D0
# download_file $bootbase/downloads/dbconfig3.D0
...
```

```
#-----------------------------------
# databases
#         database              dbconfig        updconfig       owner
#         --------              --------        ---------       -----
create_db /usr/products/upsdb   dbconfig1.D0    updconfig       products
create_db /d0dist/dist/upsdb    dbconfig2.D0    updconfig       products
create_db /d0usr/products/upsdb /tmp/dbconfig updconfig  products
```

13.3 Running the Bootstrap Procedure

To run the bootstrap, invoke the `stage1.sh[bat]` script and give it your configuration file as an argument (as shown for both UNIX and NT below). The `stage1` script downloads the bootstrap tar file and unwinds it, then runs a `stage2` script that first verifies the integrity of the configuration script and then executes it.

If `stage2` finds errors, it outputs the information to a log and tells you how to reinvoke the `stage2` script.[1] This allows you to restart the bootstrap process where you left off.

13.3.1 UNIX

To run the bootstrap, issue the command (from any directory):

```
% sh stage1.sh <config-file-name>
```

and the install will either take place, as the following output shows:

```
0% complete
(several minutes pass, the percentage updates...)
100% complete
Bootstrap succeeded.
```

or tell you of any impediments. For example:

```
The ups bootstrap cannot proceed because:
  * ups products database area already exists under /local/ups
  * ups setups scripts already exist under /usr/local/etc
  * ups setups scripts already exist under /local/ups/etc
These directories must be cleared before we can proceed.
```

If you get error messages, and you want to proceed anyway, you can run:

```
% sh stage1.sh -F <config-file-name>
```

to force the install (but we do not recommend it).

13.3.2 NT

In a DOS command prompt window, issue the command:

```
U:\> stage1.bat <config-file-name>
```

---

1. The log file is maintained as `$TMPDIR/bootups.log`, where $TMPDIR defaults to `/var/tmp` on UNIX and `%TEMP%\` on NT. The message will be at the end of the (rather long) log file.

If the install succeeds, you will see output like this:

```
Redirecting output to C:\TEMP\bootups.log
(window pops up showing percentage done)
Bootstrap succeeded!
```

A `Cygwin<version>.bat` file appears on your desktop that you can use to start **CYGWIN**.

If the install fails, it should provide error messages, for example:

```
"The bootstrap cannot proceed because:"
 * ups products database area already exists under C:\D0RunII\d0usr\products
 * ups products database area already exists under C:\D0RunII\d0usr\products
 * ups products database area already exists under C:\D0RunII\d0dist\dist
"These directories must be cleaned out before the bootstrap can run"
```

If you get error messages, and you want to proceed anyway, you can run:

`U:\> stage1.bat -F <config-file-name>`

to force the install (again, not recommended).

Chapter 14: Automatic UPS Product Startup and Shutdown

This chapter covers configuring your system to support automatic startup and shutdown of **UPS** products, and installing individual **UPS** product instances to start and stop automatically. The current bootstrap procedure (see Chapter 13: *Bootstrapping CoreFUE*) ensures that when **UPS** gets installed on a system, it is configured to enable this feature.

Note that very few products need to be run automatically; a couple of examples are **juke** and **apache**.

14.1 Configuring Your Machine to Allow Automatic Startup/Shutdown

Two scripts are supplied by **UPS** and used to run products automatically at boot time, `ups_startup` and `ups_shutdown`. A third script, called `ups`, must be supplied by you and placed in the `init.d` directory where it will be executed at boot time. It is used to configure your machine for automatic startup/shutdown and to call the first two scripts. When **UPS** gets installed and configured on a system, `ups_startup` and `ups_shutdown` get copied into separate directories under `$PRODUCTS/.upsfiles`, as follows:

```
$PRODUCTS/.upsfiles/startup/ups_startup
$PRODUCTS/.upsfiles/shutdown/ups_shutdown
```

We encourage you to use the sample `ups` script given below, with no changes except the database path (defined by `upsdb`). It works for all supported UNIX flavors.

The ups Script Particulars

The script must be called `ups`. The location of the `init.d` directory in which it must reside is OS-specific, as follows:

Operating System	Directory
IRIX, SunOS	`/etc/init.d`
Linux	`/etc/rc.d/init.d`
OSF1[a]	`/sbin/init.d`

a. On OSF1 systems the system start-up directories `init.d`, `rc2.d` and `rc0.d` are under `/sbin`, not `/etc`.

Set the `ups` file ownership and permissions properly by running:

```
% chown 0 ups
% chgrp 0 ups
```

The ups Script Contents

```
#!/bin/sh

upsdb=/local/ups/db
state=$1
case $state in
  'start')
      start=$upsdb/.upsfiles/startup/ups_startup
      (while [ ! -f $start ]; do sleep 5; done; $start) &
      ;;
  'stop')
     $upsdb/.upsfiles/shutdown/ups_shutdown
     ;;
  'config')
     case $0 in
     /*) initd=$0;;
     *)  initd='pwd'/$0;;
     esac
     sfile='echo $initd | sed -e 's;init.d/;rc3.d/S99;''
     kfile='echo $initd | sed -e 's;init.d/;rc0.d/K01;''
     ln -s $initd $sfile
     ln -s $initd $kfile
     sfile='echo $initd | sed -e 's;init.d/;rc5.d/S99;''
     kfile='echo $initd | sed -e 's;init.d/;rc6.d/K01;''
     ln -s $initd $sfile
     ln -s $initd $kfile
     ;;
  *)
     echo "usage: $0 {start|stop|config}"
     ;;
esac
```

14.2 Installing a UPS Product to Start and/or Stop Automatically

This section contains the information you need in order to install appropriate **UPS** products to run automatically. A rudimentary understanding of actions and functions in table files is helpful (see Chapter 33: *Actions and ACTION Keyword Values* and Chapter 34: *Functions used in Actions*). The autostart/autostop processes are run via a set of control files and the commands **`ups start`** and **`ups stop`**.

14.2.1 Determine if Auto Start/Stop Feature is Enabled

Unless you're installing the product **UPS** itself, you don't need to understand how the automatic startup/shutdown feature gets enabled, but you may need to determine whether it is enabled or not. The files `$PRODUCTS/.upsfiles/startup/ups_startup` and

`$PRODUCTS/.upsfiles/shutdown/ups_shutdown` are the scripts that initiate the startup/shutdown functions in **UPS**. The automatic startup/shutdown feature is enabled if and only if these three conditions are met:

- the `ups_startup` and `ups_shutdown` files exist
- the appropriate system startup files on your machine are configured to call these files (described in section 14.1 *Configuring Your Machine to Allow Automatic Startup/Shutdown*)
- an appropriate control file exists

14.2.2 Determine if Product is Appropriate for Autostart

Products that are appropriate to run in this fashion (should) come equipped with START and STOP actions in their table files. For products configured in the old **UPS** style (prior to v4), the functions comprising these actions will probably be **`sourceRequired(/path/to/<start_script>)`** and **`sourceRequired(/path/to/<stop_script>)`**, respectively. This function is described in section 34.3.27 *sourceRequired*. The specified paths must point to executables that contain start and stop instructions for the product.

Often these products also come with an TAILOR action in the table file (see section 3.6.2 *Tailoring a Product*). Once the product has been configured and tailored properly, ACTION=START functions are run at boot time to start the product and ACTION=STOP functions are run at system shutdown to stop it.

14.2.3 Edit Control File(s)

In order to make known to the system that your product is to be started at boot time, you will need to add a specific line of text to the appropriate control file. This line provides the actual start command for the product. Go to the `$PRODUCTS/.upsfiles/startup` directory[1]. There you may find one or more files with the name `<node>.products` (where `<node>` is one of the nodes in your cluster, e.g., `fsgi02.products`) and/or `<flavor>.products` (where `<flavor>` is one of the flavors in your cluster, e.g., `IRIX+5.products`). If you want your product to run as an automatic startup process on a single node in your cluster, edit the file appropriate to that node, or create the file if it doesn't exist. If you want it to run on all nodes of a particular flavor, edit or create the file appropriate to that flavor. A line for a particular product can exist in more than one file, i.e., both the corresponding node-specific and the flavor-specific files. The system runs all these files at boot time and uses only the first **`ups start`** command it finds for a product.

Add a line of the following format to the appropriate file(s):

```
/bin/su - <login_id> -c "ups start [<options>] <product>
  [<version>]"
```

Notes:

- The login id is that under which **`ups start`** must be run. If the login id is *root*, you can leave out the portion **`/bin/su - <login_id> -c`**, and the line can start from **`ups start`** (no quotes needed in this case).

1. If $PRODUCTS includes more than one database, use the `startup` directory in the database in which the active (usually current) instance of **UPS** resides.

- The **`-c`** shown here does not refer to the current chain, rather it is an option to the **`/bin/su`** call which tells it to execute **`ups start`**.

- The `ups start` script should initialize the **UPS** environment (see section 1.7.1 *Initializing the UPS Environment*), and the login id used here should not. You could opt to have the login id initialize the environment instead, but we recommend against it, especially if it's the *root* account. If the *root* account runs the standard Fermi files, then you can't use it to get into a system where there are problems with the Fermi/**UPS** environment.

In the directory `$PRODUCTS/.upsfiles/shutdown` you will find files named similarly to the startup control files. The files here tell your system to stop the process at shutdown. You will need to edit one or more of them. Add a line of the following format to the appropriate file(s):

```
/bin/su - <login_id> -c "ups stop [<options>] <product>
  [<version>]"
```

The same notes regarding the login id and **`-c`** option apply.

Make sure the permissions of all the `<node>.products` and `<flavor>.products` files are set to 744. This ensures that the files will be executable by *root* and that they have appropriate permissions for avoidance of security holes. If your products area is NFS mounted to all the appropriate machines (i.e., common to them), you only need to create these files once. If not, you need to create these files once for each products area.

14.2.4 Summary

1) Declare the product to the **UPS** database (if configuration via **`ups configure`** is required it gets done in this step by default; see section 3.6.1 *Configuring a Product*).

2) Tailor the product (usually but not always required).

3) Add a **`ups start`** control line to the appropriate file(s) in the `$PRODUCTS/.upsfiles/startup` directory.

4) Add a **`ups stop`** control line to the appropriate file(s) in the `$PRODUCTS/.upsfiles/shutdown` directory.

When the system is restarted, your process should start running on the nodes you've designated.

14.3 Disabling UPS Automatic Start/Stop of Processes

For a Single Product

To disable automatic start and stop for a single product, just remove or comment out the corresponding lines in the `<node>.products` and `<flavor>.products` files.

Disable Feature in UPS

To disable the **UPS** automatic start and stop mechanism at boot time and shutdown:

- Remove (or rename) the file `ups_startup` from the `$PRODUCTS/.upsfiles/startup` directory and the file `ups_shutdown` from the `$PRODUCTS/.upsfiles/shutdown` directory.
- Remove (or rename) all links as set in the `ups` script (e.g., `/etc/rc2.d/S99ups` and `/etc/rc0.d/K01ups` for IRIX) to the OS-specific script in the system boot area that calls the `ups_startup` and `ups_shutdown` scripts.
- Edit, remove or rename the above-mentioned OS-specific script (e.g., `/etc/init.d/ups` for IRIX).
- Empty, remove or rename the `$PRODUCTS/.upsfiles/startup` and `$PRODUCTS/.upsfiles/shutdown` directories.

14.4 A Summary of the UPS Automatic Start-up Process

Since so many different files and directories have similar names, it can be difficult to keep track of the role each plays. The process which takes place at system start-up when automatic start and stop are enabled can be summarized as follows:

1) At boot time, the link in the system's start-up area (e.g., `/etc/rc3.d/S99ups` for SunOS) points to the `ups` file (in the directory appropriate for the flavor) which runs `/path/to/ups_database/.upsfiles/startup/ups_startup`.

2) `$PRODUCTS/.upsfiles/startup/ups_startup` runs the appropriate `$PRODUCTS/.upsfiles/startup/<node>.products` and/or `$PRODUCTS/.upsfiles/startup/<flavor>.products`.

   It also runs `setups.[c]sh` so that appropriate environment variables and aliases get set.

3) The `<node>` or `<flavor>.products` file in turn runs the **`ups start`** command.

4) **`ups start`** executes the START action in the table file of the product.

The process for automatic stop is similar.

Part V Distribution Node Maintainer's Guide

Chapter 20: *Product Distribution Server Configuration*

This chapter describes how to configure and manage a **UPS** product distribution node. It was written with the assumption that the reader who is setting up a distribution server has appropriate system privileges and sufficient administrative experience to create accounts, change network services configurations, and so on.

Chapter 21: *Configuration of the fnkits Product Distribution Node*

This chapter describes the **UPS**/**UPD** configuration on the Computing Division's central product distribution node, *fnkits.fnal.gov.* Information is provided for both the `KITS` distribution database and the server's local database.

Chapter 20: Product Distribution Server Configuration

This chapter describes how to configure and manage a **UPS** product distribution node. It was written with the assumption that the reader who is setting up a distribution server has appropriate system privileges and sufficient administrative experience to create accounts, change network services configurations, and so on.

The Computing Division's primary distribution node at Fermilab is *fnkits.fnal.gov.* It is used in examples throughout this chapter, but the chapter is intended to be a general reference, and the specifics of the *fnkits* configuration are detailed in Chapter 21: *Configuration of the fnkits Product Distribution Node*.

Distribution servers like *fnkits.fnal.gov* provide a convenient central repository for product installations, but setting them up properly takes a bit of effort. An **FTP** server, a Web server, and the **UPD** configuration file (described in Chapter 31: *The UPD Configuration File*) on the server need to work together to create the right environment. This is especially important if restricted access to certain products is needed (e.g., proprietary products, or products that can only be distributed to particular systems or domains). In addition, various guidelines need to be followed in order to maintain security and keep unauthorized users from gaining control of your distribution server.

We begin the chapter by presenting step-by-step sequences of how a product distribution server responds to the two most common **UPD** commands. This discussion is intended to help you understand how all the elements of a distribution server work together to execute these and other **UPD** commands. We hope that it helps put in context the material in the remainder of the chapter, which consists mostly of administration and configuration issues.

20.1 How A Server Responds to a UPD Client Command

The two commands that a distribution server receives most frequently are **`upd addproduct`** and **`upd install`**, used to add products *to* the distribution database and to download products *from* the distribution database, respectively. Here we present step-by-step sequences of how these two processes work. As you read through the sequences of actions that follow, pay attention to which program is taking each action.

20.1.1 The Process for upd addproduct

The **`upd addproduct`** command is used to upload a product to a **UPS** product database on a distribution server. It operates by making a series of network connections to the server. All calls are made from the client system to the distribution server, who reports back results on the same data channel:

- The Web server on the distribution node is called, and a script called `ups.cgi` is used to determine if the specified product instance already exists on the distribution node. If it exists, **UPD** on the client machine prints an error and exits. If it doesn't exist, the process continues.
- The anonymous **FTP** server on the distribution node is called, and the product tar file (if any) is transferred from the user node into `/incoming`.
- The Web server is called, and `upd.cgi` is used to call `upd move_archive_file`. This script makes a product directory for the instance on the distribution node (as defined by the distribution node's `updconfig` file), installs the tar file as `${UPS_PROD_DIR}.tar` (or `${UPS_PROD_DIR}.tar.gz` or `${UPS_PROD_DIR}.zip`, according to its suffix), and unwinds part of the tar file (to make the `README` file and the `ups` and `man` directories available, if present).
- The script `upd.cgi` reports back the database, product directory, and tar file location to the client **`upd addproduct`** command.
- The anonymous **FTP** server is called, and the product's `ups` directory tar file is uploaded to `/incoming`. (If the user specified a `ups` directory, it gets uploaded over the one that was unwound from the tar file.)
- The Web server is called, and `upd.cgi` is used to call `upd moved_ups_dir`. This script makes a `ups` directory on the distribution node for the product (as defined by the `updconfig` file) and unwinds the `ups` directory tar file.
- The script `upd.cgi` reports back the database and `ups` directory to the client **`upd addproduct`** command.
- The **FTP** server and Web server are similarly called to install the table file.
- Finally, the Web server is called and `ups-decl.cgi` is used to declare the product into the distribution database.

A subset of these steps is performed to execute **`upd modproduct`** or to add a product that has a subset of these elements (e.g., one that does not include a `ups` directory).

20.1.2 The Process for upd install

The **`upd install`** command is used to download a product from a distribution node **UPS** database to a user machine. As for **`upd addproduct`**, all network connections come from the client system running **`upd install`** to the distribution server who reports any results back along that connection.

- The Web server on the distribution node is called, and `ups.cgi` is used to determine if the product instance in question exists on the server, what its dependencies are. A call to the local **UPS** determines whether the product and its dependencies exist on the user node. For the product itself and for each dependency not found on the user node, the remaining steps are taken:
- The Web server is called, and `ups.cgi` is used to determine particular details of the product on the distribution node (e.g., archive file location, product root directory, `ups` directory, and so on). If no archive file location is given, **UPD** manufactures a tar file

that should work, assuming the **FTP** server can make a tar file of directories on the fly.[1] The tar file gets named according to the convention:
`ftp://host/$UPS_PROD_DIR/..tar` (a "`.`" for the path, followed by "`.tar`").

- The **FTP** server is used to transfer and unwind the archive file, an archive of the `ups` directory, and the table file for the product.
- **UPD** declares the product on the local system.

20.2 Accounts Required for Distribution Server

A minimum of three separate user accounts are required for managing a distribution server. One of the accounts can be the normal userid of the person maintaining the configuration of the system. The three accounts needed are:

- an account under which the Web server cgi scripts will run, and which will own the products on the distribution server; usually set to *updadmin*
- an account under which the anonymous **FTP** server will run; usually set to *ftp*
- an account which can configure the administrative files for the Web server and the **FTP** server, a suggested name is *wwwadm* (can be any account, e.g., the maintainer's usual account)

Each of these accounts has particular needs and functions, and for security reasons they should be distinct from one another, as described in the following sections.

20.2.1 The updadmin Account

The *updadmin* account (which owns the cgi scripts and the products in the distribution database) has the fewest requirements. It should be usable by anyone needing to perform administrative functions related to the distribution node's **UPS** database. It should be able to schedule **cron** jobs to perform log file cleanup, reporting, and so on. It needs write access to the distribution database, the products area, and the Web server log area.

This account should *not* have write access to any of the Web server or **FTP** server configuration files.

20.2.2 The ftp Account

The *ftp* account is the home of the anonymous **FTP** service, and thus has the most restrictions on it.

The location of the *ftp* account's home directory is an important decision. The distribution node **UPS** database needs to be a subdirectory of `~ftp`, as do all the product roots and tar files for products that are to be distributed. Very often, then, `~ftp` is a whole separate file system.

1. A **WU-FTP** compatible **FTP** server is used to make tar files "on the fly".

Since this account hosts the anonymous **FTP** service, several security issues are of critical importance for setting it up securely. They are summarized here from the on-line document `http://www.cert.org/ftp/tech_tips/anonymous_ftp_config`:

- The home directory `~ftp` should not be owned by the *ftp* account. In fact, nothing whatsoever should be owned by this account. For a **UPD** server configuration, *updadmin* or perhaps *root*, would be an appropriate owner of `~ftp`.
- For the command **`upd addproduct`** to work, there must be a `~ftp/incoming` directory, writable but not readable by the *ftp* account. This directory must be readable by the *updadmin* account, however. We recommend having it owned by *updadmin* and set to mode 733.
- The anonymous **FTP** area `~ftp` needs `~ftp/etc/passwd` and `~ftp/etc/groups` files. These files should not be copies of the real system and group files. They should instead contain only the userids and groups of the files that will be encountered in the **FTP** area (`~ftp`), and should of course contain no passwords.
- The `~ftp/bin` area should contain only **ls, tar, gzip**, and **gunzip**. The `~ftp/usr/lib` area needs sufficient shared libraries to let these run. You can use **`chroot`** to test that the command runs (as in **`chroot ~ftp /bin/ls -l`** which will run the **`~ftp/bin/ls`** command under the "**chroot**ed" environment in which the **FTP** server will be living).

The **FTP** home area will be accessed via two separate avenues. The Web server will access it via its full pathname, `~ftp`, but the **FTP** server accesses this area via a **`chroot`** command. Because of these different access methods, the *ftp* account needs some symbolic links such that something chrooted to `~ftp` still finds files if the expanded `~ftp` pathname is used. For example, if `~ftp` is `/home/ftp`, then you should have symbolic links for both directory components: `home` and `ftp`. For example, when you run

```
% ls -l ~ftp
```

you should see output that contains:

```
ftp -> .
home -> .
...
```

You can create this by executing the commands:

```
% ln -s . ftp
```

```
% ln -s . home
```

This is an example of arranging things so that the **FTP** server and the Web server get a consistent view of the world, even though one uses **chroot** and the other one doesn't.

20.2.3 The wwwadm Account

This account has control of the configuration files for the **FTP** and Web servers. We refer to this account as *wwwadm* throughout this document, although this particular name is not required. Any account can be used for this, including the regular login account of the distribution server administrator, or even *root*. Similarly, a UNIX group could be created, and people in that group could be granted access to the configuration files.

The person working under this account could seriously affect the security of the distribution server by misconfiguring either of these services, therefore we recommend that access be tightly controlled.

20.3 Web Server Configuration

The Web server on the distribution node is used for two purposes:

- to run queries on the distribution node **UPS** database(s)
- to request that new products added to the server be filed away and declared

It may of course also be acting as a more general purpose Web server, however this makes the configuration somewhat more complex (the environment for execution of the cgi scripts needed for the distribution node activities may need to be different than the environment for the other activities of the server).

If it is only performing tasks related to distribution node activities, it is reasonable for the Web server to run directly as the *updadmin* account. Then all of its cgi scripts, etc. will be run as that account. If, on the other hand, other unrelated tasks are being performed on the Web server, steps should be taken to ensure that the **UPD** cgi scripts get executed as the *updadmin* account, while other activities are performed under whatever account is appropriate for them. Configuring the Web server to handle other tasks as well as running the **UPS**/**UPD** cgi scripts is beyond the scope of this document.

We recommend you use the **apache** product for your Web server, which you can install using **`upd install apache`** and tailor with **`ups tailor apache`**. Tailor it such that it runs as the account *updadmin*, and the configuration files are owned by the administrative account *wwwadm*.

20.3.1 The cgi Scripts Used to Access Distribution Database

Three **UPS**/**UPD**-related cgi scripts must reside in the Web server's `/cgi-bin` area: `ups.cgi`, `upd.cgi` and `ups-decl.cgi`. These scripts get called by some **UPD** commands, and perform the following functions:

`ups.cgi`	determines if the product exists on the server, and what its dependencies are
`upd.cgi`	installs the product on the distribution node
`ups-decl.cgi`	declares the product into the distribution database

Access to the scripts must be restricted in order to maintain control over who can run these **UPD** commands that affect the distribution database.

These scripts are provided in the **UPD** product itself, in `$UPD_DIR/cgi-bin`. We recommend that you make the Web server's versions of these files symbolic links to the `$UPD_DIR/cgi-bin` versions.

Note that when new versions of **UPD** are installed on the server, the Web server's versions of these files need to be manually updated! This does not happen automatically because the **UPD** product doesn't know where the **apache** product has the `cgi-bin` areas for its respective products.

20.3.2 Restricting Access to Distribution Database

It is critical to maintain control over the distribution database. In order to protect the database, access to the **UPD** commands must be restricted. This is done by restricting access to the Web server's cgi scripts, especially `upd.cgi` and `ups-decl.cgi` which add, modify, and delete products on the distribution server when called by the **UPD** commands.

Host-Based Access Restriction

To limit access to these scripts, we recommend using configuration entries like the following in your Web access file `./conf/access.conf` (relative to the Web server directory):

```
<Location /cgi-bin/ups.cgi>
order deny,allow
deny from all
allow from .fnal.gov
</Location>

<Location ~ /cgi-bin/up(d|s-decl).cgi>
order deny,allow
deny from all
allow from add.products.host.1
allow from add.products.host.2
</Location>
```

This example allows only hosts from the *fnal.gov* domain to execute `ups.cgi` (thus restricting product downloads), and only the (fictional) hosts *add.products.host.1* and *add.products.host.2* to run the `upd.cgi` and `ups-decl.cgi` scripts (thus restricting uploads/changes to the distribution database).

Restricting Access By User-Based Authentication

It is possible to set up user-based authentication. If the Web server prompts **UPD** for userid and password, it is configured to give the login name *${USER}* and the password *${USER}@'hostname'*. User-based authentication can be set up in a number of ways. One way is to include text like the following in your Web access file `./conf/access.conf`:

```
<Location ~ /cgi-bin/up(d|s-decl).cgi>
Anonymous_NoUserId off
Anonymous_Authoritative on
Anonymous_MustGiveEmail on
AuthUserFile /dev/null
AuthName kitstest
AuthType basic
order deny,allow
deny from all
Anonymous maint1 maint2 maint3
allow from add.products.host.1
allow from add.products.host.2
allow from add.products.host.3
# lots more of these...
require valid-user
</Location>
```

This allows access to users *maint1*, *maint2* and *maint3* from the hosts listed. The `require valid-user` line checks that the password is of the form *${USER}@'hostname'*.

In the future we hope to use kerberos-based authentication.

20.3.3 Prerequisites for Modifying the Distribution Database

There are a few prerequisites in order for the Web server's cgi scripts to run:

- **UPD** must be setup (the **apache** product in `KITS` has its scripts configured to setup **UPS**, **perl** and **python** when launching a Web server).
- The variable $PRODUCTS must be set to the database list for product distribution.

UPS needs to be setup because the `ups.cgi` script performs **UPS** commands, and to do so, the script must be able to determine the database using the $PRODUCTS path. **UPD** must be setup so that the cgi scripts can find $UPD_DIR and the associated $UPD_USERCODE_DIR to find the **UPD** configuration.

To ensure that these things are done for the *fnkits* Web server, we have added the following lines to its (**apache**) `admin/start` script:

```
#------------------------------------------------
# added for upd server, start upd, set products
umask 002
setup upd
PRODUCTS=/ftp/upsdb; export PRODUCTS
#------------------------------------------------
```

20.3.4 Permissions on Files Created in the Distribution Database

Notice that the text in the start script for *fnkits* shown in section 20.3.3 *Prerequisites for Modifying the Distribution Database* sets the **umask** with which the cgi scripts will be run. This affects the permissions on all the files generated by any cgi script run by the Web server; in particular, files created in the distribution **UPS** database, product tar files and table files, and so on.

If a tighter **umask** than `002` is used, it tends to "turn off" permissions in **UPS** product directories, which are then not appropriately group-writable when installed on end user systems.

20.4 FTP Server Configuration

For the **FTP** server on your distribution node, we recommend the one in the **wu_ftpd** product, which is available from *fnkits*. This product expects to find its configuration files in `/etc/ftpd`. This directory needs to be made writable by your *wwwadm* account, or equivalent, and to be configured to grant access to the same groups/individuals as for your Web server. We recommend that you configure the `/etc/ftpd/ftpaccess` file as shown (explanations follow the file listing):

```
class   local   real,anonymous  *.fnal.gov
class registeredhost anonymous registered.host.1
# ... lots more of these

# --------------------------------
limit   local 100   Any
log commands anonymous,real
log transfers anonymous,real inbound,outbound
chmod no anonymous
delete no anonymous
overwrite no anonymous
rename no anonymous
umask no anonymous

# anybody can do tar and compression
compress        yes      *
tar             yes      *

upload /ftp *           no
upload /ftp /incoming   yes updadmin upd 0640 nodirs

private yes

autogroup upd local
autogroup upd registeredhost

message /etc/welcome.msg        login
message /etc/upd.msg            login   local gupd registeredhost
message /etc/upd-was.mesg       login   wasregistered
message /etc/non-upd.msg        login   all

# ------------------------
class   remote  real,anonymous  *
```

This configuration accomplishes the following things:

- specifies several classes of users with `class` directives for `local`, `registered host`, and (at the end) `remote`; the `local` and `registeredhost` classes get mapped to the `upd` group in the `autogroup` lines.
- turns on logging with `log` directives
- restricts anonymous **FTP** users from doing anything dangerous, via `chmod delete`, `overwrite rename` and `umask` directives
- allows compression, and "tarring" of files with the `tar` and `compress` directives
- only allows `upload`s into `/incoming` under the `ftp` area (Note that this is redundant given the file permissions, but redundancy is sometimes good!)
- specifies different login messages with the `message` directive to let users coming in directly by **FTP** know what they're allowed to download.

Don't forget to recheck the account setup issues for the *ftp* account in section 20.2.2 *The ftp Account*.

The other **FTP** configuration files should be empty initially, except for `ftpconversions`; where the stock file from `$WU_FTPD_DIR/examples` should be sufficient. You can add entries to `ftpgroups` later to implement proprietary-style access control on some products, if needed.

The **FTP** configuration files should be writable by the *wwwadm* account.

20.5 UPD Configuration Items

There are several **UPD** configuration items in the `${UPD_USERCODE_DIR}/updconfig` file that are used exclusively on product distribution servers. These must be set properly in order to have a working install server.

20.5.1 Archive File Keywords and ${SUFFIX}

The special keywords are:

UNWIND_ARCHIVE_FILE	the absolute path to the archive file (not unwound)
UPS_ARCHIVE_FILE	the path that **UPD** uses to declare the archive file to **UPS** (minus `ftp://<host>` which gets prepended before it is declared[1])

UNWIND_ARCHIVE_FILE and UPS_ARCHIVE_FILE are similar to other **UPD** configuration file variable pairs (see section 31.3.1 *Required Locations*). The values for both these variables are paths that must end in the file name appended by ${SUFFIX}. ${SUFFIX} is a read-only variable that describes the type of archive (e.g., `tar`, `tar.gz`, `zip`). Its value comes from the **UPD** command line. Including ${SUFFIX} on the end of the definition is *mandatory*; **UPD** cannot install a product whose archive file does not end in the proper suffix.[2]

Examples

Here are sample definitions of UNWIND_ARCHIVE_FILE and UPS_ARCHIVE_FILE:

```
UNWIND_ARCHIVE_FILE="/ftp/archives/${UPS_PROD_NAME}${UPS_PROD_VERSION}${UPS_PROD_FLAVOR}${UPS_PROD_QUALIFIERS}.${SUFFIX}"
```
(*all on one line in real file*)

```
UPS_ARCHIVE_FILE="${UNWIND_ARCHIVE_FILE}"
```

UPD will then use the following path to declare the product tar file to **UPS**:

```
ftp://<host>/ftp/archives/${UPS_PROD_NAME}${UPS_PROD_VERSION}${UPS_PROD_FLAVOR}${UPS_PROD_QUALIFIERS}.${SUFFIX}
```

To make **UPD** use an **FTP** server at a particular port number (e.g., 777), define UPS_ARCHIVE_FILE as:

```
UPS_ARCHIVE_FILE=":777${UNWIND_ARCHIVE_FILE}"
```

1. **`<host>`** is the current **`-h <host>`** argument to **`upd addproduct`**. It defaults to *fnkits.fnal.gov.*
2. You could in principle use a specific suffix, e.g., ".tar" in place of ${SUFFIX} if the actions in the file don't repack or compress/decompress the files.

20.5.2 Pre- and Postdeclare ACTIONs

As with any `updconfig` file, you can define pre- and/or postdeclare actions. These are described in section 31.4 *Pre- and Postdeclare Actions*. Briefly, they define actions for **UPD** to take just before or just after declaring a product to the database. They can be used for a number of tasks on a distribution server, e.g.,

- to apply permission file changes
- to add symbolic links
- to update html index files

In particular, when combined with use of the **optionlist** product, described in section 20.6.5 *Flagging Special Category Products Using Optionlist*, you can cause certain products to use a stanza in the configuration file that sets special group access permissions on the files that have just been installed. For example, the following text shows a predeclare action that makes the files for some product on *fnkits* readable only by group FNALONLY, which in combination with the **FTP** server configuration, means that only users in the `.fnal.gov` domain can access those files:

```
action = predeclare
    #
    # fix group permissions
    #
    Execute("chgrp FNALONLY ${UNWIND_TABLE_DIR}/*", NO_UPS_ENV)
    Execute("chmod o-rwx ${UNWIND_TABLE_DIR}/*", NO_UPS_ENV)
    Execute("chmod a+r ${UNWIND_TABLE_DIR}/*.table", NO_UPS_ENV)
```

20.6 Administrative Tasks and Utilities

20.6.1 Reporting FTP and Web Server Activity Using Ftpweblog

For reporting, we recommend the **ftpweblog** product, which allows you to build reports combining **FTP** and Web accesses. The **apache** product's tailor script writes a statistics script, `monthlystats`, that can be modified to include calls to **ftpweblog** to request the **FTP** transfers for a **UPD** server machine. Then, whenever `monthlystats` gets run, you'll generate a combined report in your Web server's log summary area.

To modify `monthlystats` to get this information, first change the list of logfiles (`loglist`) to include the **FTP** `xferlog` file. Change

```
loglist="$accesslog $errorlog $agentlog $referlog"
```

to

```
loglist="$accesslog $errorlog $agentlog $referlog /var/log/ftpd/xferlog"
```

Next, just after the call to **ftpweblog** that's already there, add an extra invocation of **ftpweblog** to include the `xferlog` data for "today" in the summary report, e.g.,

```
...
mv ${thismonth} ${thismonth}.bak
totals="${thismonth}.bak"
ftpweblog                   \
-N "${title}"               \
-i ${totals}                \
-t ftp /var/log/ftpd/xferlog \
 > ${thismonth}
...
```

20.6.2 Restricting Access for Uploads to Distribution Database

The permissions on the Web server cgi scripts control access to the **`upd add/mod/delproduct`** commands which modify the distribution database. In section 20.3.2 *Restricting Access to Distribution Database* we show how to modify the Web server's `access.conf` file to allow the appropriate hosts access to `upd.cgi` and `ups-decl.cgi`. After editing the file, restart the Web server on the distribution node. To do this, you can enter these commands:

```
% su updadmin
    password: (enter password)
% ups restart apache
% exit
```

20.6.3 Restricting Access for Downloads from Distribution Database

In order to register particular hosts for downloading products, you need to add their hostnames to access files for both the Web and the **FTP** servers, as described in sections 20.3.2 *Restricting Access to Distribution Database* and 20.6.4 *Restricting Distribution of Particular Products*, respectively. The **FTP** access file is maintained at `/etc/ftpd/ftpaccess`, and the Web access file is found relative to the Web server directory at `./conf/access.conf`.

On *fnkits* we use the **`cmd addkits`** script to add hosts to the appropriate files. This script can be found in the `$UPD_DIR/admin` directory.

20.6.4 Restricting Distribution of Particular Products

The best available mechanism for limiting distribution of particular products is via group ids, using the **FTP** server's ability to map particular classes of clients to particular groups. By making a set of files under `~ftp` readable by a particular group only, the **FTP** server automatically restricts access to those files, allowing access only to those clients which are mapped to that group. For example, on *fnkits.fnal.gov* there is a group for registered users, a separate group for each proprietary product, and a group for on-site-only access. Groups can be created as you need them.

Note that if you do create such groups, you must either include the *updadmin* account in each group so that it has permission to change files to these groups (via **`chgrp`**), or use a mechanism like **`cmd`** or **`sudo`** to allow the *updadmin* account to do this.

20.6.5 Flagging Special Category Products Using Optionlist

UPD supports creation and use of a special table-file-only product called **optionlist** on a **UPD** server. In this table file, you can define options specific to products which may subsequently get installed on the server[1]. **UPD** checks this table file automatically when executing **`upd install`**, **`upd addproduct`**, or **`upd modproduct`**. **optionlist** provides a check in case a product provider forgets to flag a product as belonging to a special category.

`upd install` only looks for "proprietary" in the options, to see if it should prompt the user for account information and do SITE GROUP commands, and so on. The **`upd addproduct`** and **`upd modproduct`** commands pass any listed option(s) over to the **`upd move_table_file`**, etc., commands on the server side, thereby setting the listed flags as if they had been put on the command line with the **`-O`** option. In other words, **UPD** effectively ignores all options except **`proprietary`**, and just passes them through to the **UPD** configuration on the server.

The **optionlist** product should be declared as version "only" and flavor "NULL". It requires user-defined keywords of the form `_UPD_OPTS_<PRODUCT>="<option_list>"` (see section 27.2 *Keywords: Information Storage Format*) defined in `updconfig`, and uses them as shown in the example below:

```
FILE=Table
Product=optionlist

group:
Flavor=ANY
Qualifiers=""

common:
# Proprietary products
_upd_opts_edt="proprietary"
_upd_opts_flint="proprietary"
end:
```

According to this table file, whenever an instance of **edt** or **flint** gets installed on the server, **`upd install`** gets run with the option **`-O proprietary`**. These products will only match a `updconfig` file stanza that specifies `options=proprietary`.

You can download the current *fnkits* **optionlist** table file for reference by issuing the command:

```
% upd fetch -J @table_file optionlist
```

This gets the file `optionlist_only_NULL.table`. There are about 80 entries in the file at the time of this writing.

1. Really early versions of **UPD** used a **proprietarylist** product for proprietary products; the process has now been generalized to include other product types.

20.6.6 Searching FTP Server Logfiles Using Searchlog

On *fnkits* we have a simple cgi script that lets users search for downloads/uploads of particular products in the **FTP** server logs. It can be run from `http://fnkits.fnal.gov/cgi-bin/searchlog.cgi`. The file content is shown here:

```
#!/bin/sh

# adapt the following to find your xferlogs if needed
#
logfiles="‘echo /var/adm/xferlog* /var/log/ftpd*/xferlog*‘"

echo "Content-type: text/html"
echo

if [ $# = 0 ]
then
    cat <<EOP
        <html> <head> <title> upd Downloads Search </title> </head>
        <body> <h1> upd Downloads Search </h1>
        Please enter a product and version
        <isindex> </body> </html>
EOP

else
    if [ $# != 2 ]
    then
        cat <<EOP
        <html> <head> <title> Invalid Search </title> </head>
        <body> <h1> Inalid search </h1>
        Please use your back button and enter a product and version!
        </body> </html>
EOP
    else
        cat <<EOP
        <html> <head> <title> Search Results </title> </head>
        <body> <h1> Search Results </h1>
        Search results for product $1 version $2
        <pre>
EOP
        for f in $logfiles
        do
          case $f in
          *.gz|*.Z) gunzip < $f ;;
          *) cat $f ;;
          esac
        done 2>/dev/null | grep "$1" | grep "$2"
        cat <<EOP
        </pre> </body> </html>
EOP
    fi
fi
```

20.7 Product Distribution via CD-ROM

A CD-ROM can be used as a distribution database. You start with a local directory tree containing the necessary files and products, and then, using appropriate tools, you create an image of this area and burn it onto a CD. The Computing Division has created images for use on Linux machines. You can obtain a CD-ROM of one of the images, use one of the images to create your own CD-ROM, or if none of the provided images meets your needs, you can create your own image and make a CD-ROM from that.

As of this writing, the procedure for creating product distribution CD-ROMs is under development. We plan to create and maintain a Web page with this information at `http://www.fnal.gov/docs/products/ups/ReferenceManual/misc/cdrom.html`.

Chapter 21: Configuration of the fnkits Product Distribution Node

This chapter describes the **UPS**/**UPD** configuration on the Computing Division's central product distribution node, *fnkits.fnal.gov.*[1] Information is provided for both the `KITS` distribution database and the server's local database.

21.1 UPS Configuration for KITS Database

The `KITS` database on the *fnkits.fnal.gov* node has a fairly minimal configuration file, typical for distribution databases:

- The database is configured to allow all registered nodes to read and use the products in it.
- Statistics are not collected for any products.
- Locations are defined for product instances, the **UPS** initialization files and the **UPD** configuration file.

For reference, we list the contents of the `dbconfig` file for `KITS` (minus the comments):

```
FILE = DBCONFIG
AUTHORIZED_NODES = *
PROD_DIR_PREFIX = /ftp/products
STATISTICS =
SETUPS_DIR = /fnal/etc
UPD_USERCODE_DIR = /fnal/ups/db/.updfiles
```

21.2 UPS Configuration for local Product Database

The local database on the *fnkits* node is maintained at `/fnal/ups/db`. The `dbconfig` file for this database is typical for databases on user nodes, where products are unwound and available for use. This file happens to contain all the information that the `dbconfig` file for the `KITS` distribution database does, except that the product area, defined by PROD_DIR_PREFIX, is different. In addition to this content, there are definitions of target directories for various product information files (e.g., man pages).

1. Other names used for this server are: *fnkits*, *kits*, *kits.fnal.gov*, *upd*, and *upd.fnal.gov.*

For reference, we list the contents of the `dbconfig` file for the local database (minus the comments):

```
FILE = DBCONFIG
AUTHORIZED_NODES = *
PROD_DIR_PREFIX = /fnal/ups
STATISTICS =
MAN_TARGET_DIR = /fnal/ups/man
CATMAN_TARGET_DIR = /fnal/ups/catman
INFO_TARGET_DIR = /fnal/ups/Info
HTML_TARGET_DIR = /fnal/ups/htmldocs
NEWS_TARGET_DIR = /fnal/ups/news
SETUPS_DIR = /fnal/etc
UPD_USERCODE_DIR = /fnal/ups/db/.updfiles
```

In particular, notice that UPD_USERCODE_DIR is set to the same value in both files. This indicates that the databases share a **UPD** configuration.

Soon after the release of this document (mid-2000), the local database on *fnkits* will point to its own **UPD** configuration.

21.3 UPD Configuration

21.3.1 updconfig File Organization

The **UPD** configuration for both the `KITS` distribution database (`/ftp/upsdb`) and the database used for locally installed products (`/fnal/ups/db`) is contained in the same file, `${UPD_USERCODE_DIR}/updconfig`.[1] ${UPD_USERCODE_DIR} is defined to be `/fnal/ups/db/.updfiles` in the `dbconfig` files for both databases.

The `dbconfig` file includes several stanzas, each of which pertains to a category of product. The product-matching criterion for each stanza is an option which indicates the category. The categories are: `default` (no option), `local`, `fermitools`, `proprietary`, `fnalonly`, and `usonly`. For example, the `GROUP:` section of the stanza for `default` products is empty, the one for `proprietary` contains the `options` line:

```
group:
     options = "proprietary"
```

and so on. The contents of the `COMMON:` sections for each category, namely the location and file name definitions and any actions, are listed in sections 21.3.4 *Location and File Name Definitions* and 21.3.5 *Pre- and Postdeclare ACTIONS*.

1. This is changing mid-2000; the local database will have a separate **UPD** configuration file.

21.3.2 The Recognized Product Categories

`default`	The `default` category is the most commonly used, and is for regular products added to the `KITS` database (`/ftp/products` or `/ftp/KITS`) for distribution to any on-site or registered off-site node[1]. The products are set to group `upd`, and group-read-only. No option is associated with the default.
`local`	(This will be dropped mid-2000.) The `local` category is for products installed (using **`upd install`**) into the local database, `ftp/ups/db`, for use on the *fnkits* node itself (as opposed to those added to `KITS` for distribution). For these products, the **`-O local`** option must be included in the **`upd install`** command. (Used only by the *fnkits* system managers.)
`fermitools`	`fermitools` products are locally-developed and supported software packages (which are not available elsewhere, generally) that we make available to the public via our FermiTools program[2]. These products are installed in the `KITS` database, are world-readable, and have a symlink hierarchy under `/ftp/pub`. The `/ftp/pub` hierarchy has been created with the same structure as `/ftp/KITS`.
`proprietary`	The `proprietary` category includes products for which Fermilab has a limited number of licenses. These products are installed in the `KITS` database, and made accessible only to special groups.
`fnalonly`	The `fnalonly` category is for products accessible only to the `fnal.gov` domain. They are installed in the `KITS` database, set to group `fnalonly`, and are group-read-only.
`usonly`	US-only (United States only) products are accessible only to U.S. government (`.gov`) and military (`.mil`) domains. In general, these are products for which distribution to other countries is illegal. They are installed in the `KITS` database, set to group `usonly`, and are group-read-only.

21.3.3 Matching Product Categories to updconfig Stanzas

When adding a product belonging to any of the categories `fermitools`, `proprietary`, `fnalonly` or `usonly`, do not specify the corresponding option via the **`-O`** flag on the **`upd addproduct`** command line. Instead, first fill out and submit the **Special UPD Product Registration** form (at `http://fnkits.fnal.gov/specialprod.html`) identifying the product and its category. After you receive a confirmation via email, add the product as you would a "default" product (see Chapter 17: *Making Products Available For Distribution*). The option corresponding to your selected category gets set automatically in order to invoke the proper stanza in the `updconfig` file.

---

1. See the **Product Distribution Platform Registration Request** form at `http://www.fnal.gov/cd/forms/upd_registration.html`.
2. For more information on FermiTools, see `http://www.fnal.gov/fermi-tools/`.

21.3.4 Location and File Name Definitions

All Product Categories (except local)

The following location and file name definitions are shared by the stanzas for the product categories `default`, `usonly`, `fnalonly`, `proprietary`, and `fermitools`:

```
UPS_THIS_DB = /"ftp/upsdb"
UNWIND_PROD_DIR="/ftp/products/${UPS_PROD_NAME}/${UPS_PROD_VERSION}/
  ${UPS_PROD_FLAVOR}/${UPS_PROD_NAME}_${UPS_PROD_VERSION}_${UPS_PROD_FLAVOR}
  ${UPS_PROD_QUALIFIERS}"
UNWIND_UPS_DIR = "${UNWIND_PROD_DIR}/ups"
UNWIND_TABLE_DIR = "/ftp/products/${UPS_PROD_NAME}/${UPS_PROD_VERSION}/
  ${UPS_PROD_ FLAVOR}"
UNWIND_ARCHIVE_FILE = "${UNWIND_PROD_DIR}.${SUFFIX}"
UPS_TABLE_FILE="${UPS_PROD_NAME}_${UPS_PROD_VERSION}_${UPS_PROD_FLAVOR}
  ${UPS_PROD_QUALIFIERS}.table"
UPS_TABLE_DIR = "${UNWIND_TABLE_DIR}"
UPS_PROD_DIR = "${UNWIND_PROD_DIR}"
UPS_UPS_DIR = "ups"
```

After the mid-2000 change, we expect a few of these values to change, as follows:

```
UNWIND_PROD_DIR="/ftp/products/${UPS_PROD_NAME}/${UPS_PROD_VERSION}/
  ${UPS_PROD_FLAVOR}/${UPS_PROD_NAME}_${UPS_PROD_VERSION}_${UPS_PROD_FLAVOR}
  ${UPS_PROD_QUALIFIERS}"
UNWIND_UPS_DIR = "${UPS_PROD_DIR}/ups"
UNWIND_TABLE_DIR = "${UPS_TABLE_DIR}"
UPS_TABLE_DIR = "/ftp/products/${UPS_PROD_NAME}/${UPS_PROD_VERSION}/
  ${UPS_PROD_FLAVOR}"
```

The UPS_ARCHIVE_FILE definition varies. Different **FTP** server port numbers with different permissions are used according to the product category. For `default`, `proprietary` and `fermitools` the value is:

```
UPS_ARCHIVE_FILE = ${UNWIND_ARCHIVE_FILE}
```

For `fnalonly`, it is:

```
UPS_ARCHIVE_FILE = ":9021${UNWIND_ARCHIVE_FILE}"
```

For `usonly`, it is:

```
UPS_ARCHIVE_FILE = ":8021${UNWIND_ARCHIVE_FILE}"
```

local

The location and file name definitions for `local` products are (again, this will be in a separate **UPD** configuration file after mid-2000):

```
UPS_THIS_DB= "/fnal/ups/db"
UNWIND_PROD_DIR=
"/fnal/ups/${UPS_PROD_NAME}/${UPS_PROD_VERSION}/${UPS_PROD_FLAVOR}${UPS_PROD_QUALIFIE
RS}"
UNWIND_UPS_DIR= "${UNWIND_PROD_DIR}/ups"
UNWIND_TABLE_DIR = "${UPS_THIS_DB}/${UPS_PROD_NAME}"
UPS_TABLE_FILE= "${UPS_PROD_VERSION}.table"
UPS_PROD_DIR = "${UNWIND_PROD_DIR}"
UPS_UPS_DIR = "ups"
```

21.3.5 Pre- and Postdeclare ACTIONS

The stanzas for all categories of product except `local` include a PREDECLARE and a POSTDECLARE action.[1]

- In each case, the PREDECLARE action includes a set of **`execute`** statements to **`chmod/chgrp`** the files to the right group id and permissions, and another set to symlink files under `/ftp/KITS` to provide the old-style (**UPS/UPD** v3) `KITS` hierarchy[1] of `KITS/Flavor/product/version`. In addition, `fermitools` includes commands to send notification email.
- The POSTDECLARE action makes a convenience tar file of the `ups` directory for users downloading via **ftp**.

The **`execute`** statements in each stanza are similar, but not identical. We first list them for the default case, and then list the differences for the other product categories relative to the default.

The stanza for `local` products contains no actions.

PREDECLARE Action for default Products

The PREDECLARE action for the `default` product category fixes group permissions:

```
action = predeclare
Execute("chgrp upd ${UNWIND_TABLE_DIR}/*", NO_UPS_ENV)
Execute("chmod o-rwx ${UNWIND_TABLE_DIR}/*", NO_UPS_ENV)
Execute("chmod a+r ${UNWIND_TABLE_DIR}/*.table", NO_UPS_ENV)
```

and makes old-`KITS` compatible hierarchy files:

```
Execute("test -d  /ftp/KITS/${UPS_BASE_FLAVOR}/${UPS_PROD_NAME}/${UPS_PROD_VERSION} ||
mkdir      -p      /ftp/KITS/${UPS_BASE_FLAVOR}/${UPS_PROD_NAME}/${UPS_PROD_VERSION}",
NO_UPS_ENV)

Execute("cd  /ftp/KITS/${UPS_BASE_FLAVOR}/${UPS_PROD_NAME}/${UPS_PROD_VERSION};  rm  -f
${UPS_PROD_NAME}_${UPS_PROD_VERSION}_${UPS_PROD_FLAVOR}${UPS_PROD_QUALIFIERS}.*",
NO_UPS_ENV)

Execute("cd         /ftp/KITS/${UPS_BASE_FLAVOR}/${UPS_PROD_NAME}/${UPS_PROD_VERSION};
/usr/bin/ln -fs ${UNWIND_PROD_DIR}.* . || true",NO_UPS_ENV)
```

PREDECLARE Action for FermiTools Products

The PREDECLARE action is the same as for the `default` products except for the changes noted here.

For `fermitools`, there is no **`Execute ("chgrp...")`** command. The first **`chmod`** command is **`o+rx`** rather than **`o-rwx`**:

```
Execute("chmod o+rx ${UNWIND_TABLE_DIR}/*", NO_UPS_ENV)
```

Also for `fermitools`, commands are included to send notification e-mail:

```
Execute("sh -c \"dir=/ftp/pub/${UPS_PROD_NAME}/${UPS_PROD_VERSION}; test -d \\\$dir ||
mkdir -p \\\$dir\"", NO_UPS_ENV)

Execute("sh   -c   \"dir=/ftp/pub/${UPS_PROD_NAME}/${UPS_PROD_VERSION};   cd   \\\$dir;
/usr/bin/ln -sf ${UNWIND_PROD_DIR}.* .; /usr/bin/ln -sf ${UNWIND_PROD_DIR}/README .\"
|| true", NO_UPS_ENV)

Execute   ("echo   Fermitools   product   ${UPS_PROD_NAME}   ${UPS_PROD_VERSION}   -f
${UPS_PROD_FLAVOR} has been added to fnkits | /bin/mail fermitools_support\@fnal.gov",
NO_UPS_ENV)
```

---

1. After mid-2000, some of the functions in the PREDECLARE actions move to a common POSTDECLARE action, namely the functions that make old-`KITS` compatible hierarchy files.

1. Currently nothing prunes old links or files from this hierarchy.

PREDECLARE Action for proprietary, fnalonly and usonly Products

The PREDECLARE action is the same as for the `default` products except for the changes noted here.

For `proprietary` products, the **`chgrp`** command changes to:

```
Execute("chgrp ‘echo ${UPS_PROD_NAME} | sed -e ’s/^vx.*/vx_dart/’ | tr a-z A-Z‘
${UNWIND_TABLE_DIR}/*", NO_UPS_ENV)
```

For `fnalonly` and `usonly` products, the group used in the **`chgrp`** command changes to `FNALONLY` and `USONLY`, respectively:

```
Execute("chgrp FNALONLY ${UNWIND_TABLE_DIR}/*", NO_UPS_ENV)
```

and

```
Execute("chgrp USONLY ${UNWIND_TABLE_DIR}/*", NO_UPS_ENV)
```

POSTDECLARE Action for All Product Categories (except local)

The POSTDECLARE action makes a `${UNWIND_PROD_DIR}.ups.tar` tar file:

```
action = postdeclare
Execute("test -d \"${UNWIND_UPS_DIR}\" && cd ${UNWIND_UPS_DIR} && tar cf
${UNWIND_PROD_DIR}.ups.tar . || true", NO_UPS_ENV)
```

21.4 fnkits Server Maintenance

21.4.1 User Accounts and Group Ids

The *fnkits* Web and **FTP** server configuration files are owned by *oss*.

The Web server runs as *updadmin*. *updadmin* owns all of the **FTP**-served product files and the `KITS` **UPS** database, and therefore also the Web server logs.

Many group ids are used, see `/etc/ftpd/ftpgroups` on *fnkits* for the complete list.

21.4.2 Database and Configuration File Locations

The `KITS` distribution database is `/ftp/upsdb`. or /ftp/products...??

The local database is under `/fnal/ups/db`.

The Web server configuration files are maintained under the `/fnal/www` directory.

The **FTP** configuration files live under `/etc/ftpd`, except for `usonly` and `fnalonly` products which are kept under `/etc/ftpd-usonly` and `/etc/ftpd-fnalonly`, respectively.

“Sanitized” versions of the configuration files (with hostnames and such trimmed) are available at `http://ftp.fnal.gov/`.

21.4.3 Web Server and FTP Log File Information

Logs are kept for roughly 30 days.

The Web server logs are kept in `/fnal/log/www`.

The **FTP** logs are kept under `/var/adm/ftpd`, except for `usonly` and `fnalonly` products which are kept under `/var/adm/ftpd-usonly` and `/var/adm/ftpd-fnalonly`, respectively.

The log summaries are maintained under `/fnal/www/kits/html/logs`, also accessible via `http://ftp.fnal.gov/logs`. They are prepared using the product **ftpweblog**.

Part VII Administrator's Reference

Chapter 27: *Information Storage Format in Database and Configuration Files*

This chapter introduces the files **UPS** uses for product management. It also describes the format of the information storage in these files, which is in the format of KEYWORD=VALUE pairs. The supported keywords are listed and described.

Chapter 28: *Version Files*

Version files are **UPS** database files that contain information specific to the local installation and declaration of the declared product instances. The contents of version files are described in this chapter.

Chapter 29: *Chain Files*

UPS/UPD supports *chains* to product versions, and chain information is maintained in chain files. In this chapter we describe chain files and how they interact with version files.

Chapter 30: *The UPS Configuration File*

A **UPS** database can be configured and customized using the file `dbconfig`, described in this chapter. It is used to define keywords which control quantities such as:

- which nodes can access products maintained in the database
- the directory under which products are installed
- which products will have usage statistics collected
- the directories for product **man** pages and **Info** files
- the directory containing the **UPS** initialization files
- the directory containing the **UPD** configuration file
- the **UPS** database version

Chapter 31: *The UPD Configuration File*

UPD can be configured and customized on your system using the file `updconfig`, described in this chapter. By providing default values for several variables (mostly product file and directory locations), the `updconfig` file controls where **UPD** installs products and miscellaneous product-related files. It can also be used to define supplementary actions for **UPD** to perform when installing or updating products.

Chapter 32: *The UPP Subscription File*

UPP is a layer on top of **UPD** that can be used to facilitate the update of products on a local **UPS** node as new versions become available on a product distribution node. **UPP** is configured on the local node by subscription files, which we describe in this chapter. The functions **UPP** can be configured to perform on a local node include:

- notify the client of new and updated products on a specified distribution node
- perform product installations and updates
- install/update product dependencies and resolve chains to maintain integrity of main product
- delete old product versions

Chapter 27: Information Storage Format in Database and Configuration Files

This chapter introduces the files **UPS/UPD** uses for database and product management. It also describes the format of the information storage in these files, which is in the form of KEYWORD=VALUE pairs. The supported keywords are listed and described.

Most of the time, product installers and **UPS** database managers can get all the information they need about a product or about the contents of a database via the **`ups list [-K <keywordList>]`** command output (described in section 22.11 *ups list*), which is fairly easy to interpret. However, it's helpful to understand the database files when dealing with complex situations. The keywords described in this chapter which appear in the database files also appear in the **`ups list -l`** output.

27.1 Overview of File Types

The information that **UPS** needs in order to configure and manage a database and to identify, locate, and retrieve product instances resides in a set of ASCII files in the **UPS** database. The information that **UPD** needs for installing products also resides there. The files used for these purposes include:

- *Version files* tell **UPS** where to find all the files associated with a particular version of a product on the local system, and contain some other information specific to the local installation of the product. They are generally named according to the scheme `vx_y.version`, e.g., `v1_0.version`. These are described in Chapter 28: *Version Files*.
- *Chain files* are optional and contain pointers to version files, thus providing convenient access to particular product versions on the local system. They are generally named according to the scheme `chainname.chain`, e.g., `current.chain`. These are described in Chapter 29: *Chain Files.*
- The *UPS database configuration file* defines things such as which nodes can access products maintained in the database, and which directories house products, **man** pages, **UPS** initialization files, the **UPD** configuration file, and so on. It is described in Chapter 30: *The UPS Configuration File*.
- The *UPD configuration file* controls where **UPD** installs products and miscellaneous product-related files. It can also be used to define supplementary actions for **UPD** to perform when installing or updating products. It is described in Chapter 31: *The UPD Configuration File*.

These files are sometimes referred to collectively as *UPS database files*. They store information in the format of *keywords*.

This information storage format is also used in *table files*, which are provided by the product developer and discussed in Part VIII *Developer's Reference*. They contain product-specific, system-independent information. Table files can be maintained in the database, but they are not constrained to reside there, and in fact usually reside under the product root directory.

27.2 Keywords: Information Storage Format

27.2.1 What is a Keyword?

UPS/UPD utilizes a set of keywords that collectively store the information **UPS/UPD** requires for managing products. A *keyword* represents a category of information used by **UPS/UPD**, it is akin to a variable.[1] A *keyword line* in a file assigns a value to a keyword in the format `KEYWORD = VALUE`.

The supported keywords are listed and described in the table in section 27.4 *List of Supported Keywords*. Some of the keywords can be used in all the file types, others are restricted to certain file types. A few keywords have default values.

Keywords and their values are **not** case-sensitive.

27.2.2 Keyword Syntax

When two or more words are used to make up one keyword, they are generally separated by an underscore (_) for readability. All the provided keywords use full words except:

DB	is used instead of DATABASE
DIR	is used instead of DIRECTORY
PROD	is used instead of PRODUCT

27.2.3 User-Defined Keywords

In addition to those listed, **UPS/UPD** allows user-defined keywords (where *user* in this context refers to a product developer or administrative user). All user-defined keywords must have underscore (_) as the initial character. While parsing, any unrecognized (i.e., user-defined) keywords are ignored by **UPS**, but they are preserved across rewrites of the files.

1. And in many cases a keyword has an associated read-only variable usable in functions in the table file and/or the `updconfig` file.

27.2.4 How UPS/UPD Sets Keyword Values

Keywords stored in the **UPS** database configuration file (described in Chapter 30: *The UPS Configuration File*) and the **UPD** configuration file (described in Chapter 31: *The UPD Configuration File*) are given values according to the configuration chosen when **UPS/UPD** was installed and configured. See Chapter 13: *Bootstrapping CoreFUE* for information on choosing values during the installation of **UPS/UPD**.

Keywords stored in version or chain files are set at the time that the corresponding product instance and/or chain is declared to the **UPS** database. Those stored in table files are usually set by the product developer. If a keyword is stored in both the database configuration file and another file, then, for the corresponding product instance(s), the value set at product or chain declaration overrides the one set in the database configuration file.

27.3 Flexibility of File Syntax

The syntax of the database files is *fixed* but *forgiving*. It is *fixed* in the sense that **UPS** commands automatically create the version and chain files in a particular **UPS**-supported format. Any **UPS** command that modifies information in these files rewrites the file to disk according to the same format. The syntax is *forgiving*, however, in that when you perform manual file updates, **UPS** will ignore blank lines and extra whitespace (spaces and tabs).

Comment lines can be placed anywhere in the file and must begin with a pound sign (#). However, if you want comments to be preserved upon rewrite, they must be the first lines in the file.

27.4 List of Supported Keywords

The following table gives information about each provided keyword. The last five columns indicate which database file the keyword may be used in. The headings D, U, C, V and T refer to:

D	Database configuration file (`dbconfig`)
U	**UPD** configuration file (`updconfig`)
C	Chain file
V	Version file
T	Table file

Keyword and Default Value (if any)	Description and Notes (if any)	D	U	C	V	T
ACTION	defines an action (described in Chapter 33: *Actions and ACTION Keyword Values*), i.e., groups together a list of functions associated with a command (e.g., ACTION=SETUP)		U			T
ARCHIVE_FILE	archive file name/location; used by **UPD**				V	
AUTHORIZED_NODES Default: All nodes (*); taken from **UPS** database configuration file	authorized nodes	D			V	
CATMAN_SOURCE_DIR Default: under the `${UPS_UPS_DIR}/toman/catman` directory	location of catman files (formatted man page files) included with instance					T
CATMAN_TARGET_DIR	directory into which catman files are to be copied	D				
CHAIN	chain name			C		
COMMON:	groups together actions that apply to all instances represented in "GROUP:"; COMMON: is only valid within a GROUP:		U			T
COMPILE_DIR	directory in which the compile file resides				V	
COMPILE_FILE	the name of the file containing compiled functions (see Chapter 37: *Use of Compile Scripts in Table Files*)				V	
DECLARED Default: current date and time	the date/time that the instance was declared to **UPS** or declared with a chain Note: often has multiple values, one for each declaration (e.g., for subsequent chain declarations)			C	V	
DECLARER Default: current user	userid of user that performed the declaration Note: often has multiple values, one for each declaration (e.g., for subsequent chain declarations)			C	V	
DESCRIPTION	product description		U	C	V	T
END:	marks the end of a "GROUP:" or "COMMON:"; one "END:" marker is used to jointly end a "GROUP:" and an included "COMMON:"		U			T

Keyword and Default Value (if any)	Description and Notes (if any)	D	U	C	V	T
FILE	type of file (possible values: DBCONFIG, UPDCONFIG, CHAIN, VERSION, TABLE)	D	U	C	V	T
FLAVOR	product instance flavor Note: To easily accommodate flavor-neutral **`setup`** functions in a table file, FLAVOR can take the value ANY, but *only* in a table file.		U	C	V	T
GROUP:	groups together multiple instances; all entries subsequent to this "GROUP:" are part of it until an "END:" marker is reached		U			T
HTML_SOURCE_DIR Default: under the `${UPS_UPS_DIR}/tohtml` directory	location of html files included with instance *not supported in UPS v4*					T
HTML_TARGET_DIR	directory into which html files are to be copied *not supported in UPS v4*	D				
INFO_SOURCE_DIR Default: under the `${UPS_UPS_DIR}/toInfo` directory	location of Info files included with instance					T
INFO_TARGET_DIR	directory into which Info files are to be copied	D				
MAN_SOURCE_DIR Default: under the `${UPS_UPS_DIR}/toman/man` directory	location of unformatted man page files included with instance					T
MAN_TARGET_DIR	directory into which formatted man pages are to be copied	D				
MODIFIED Default: Current date/time	last time the associated instance was changed Note: often has multiple values, one for each declaration/modification (e.g., for subsequent chain declarations)			C	V	
MODIFIER Default: Current user	userid of user that modified the instance Note: often has multiple values, one for each declaration/modification (e.g., for subsequent chain declarations)			C	V	

Keyword and Default Value (if any)	Description and Notes (if any)	D	U	C	V	T
NEWS_SOURCE_DIR Default: under the `${UPS_UPS_DIR}/tonews` directory	location of news files included with instance *not supported in UPS v4*					T
NEWS_TARGET_DIR	directory into which news files are to be copied (for posting to a newsgroup) *not supported in UPS v4*	D				
ORIGIN	master source file; see option **`-D`** in Chapter 24: *Generic Command Option Descriptions*				V	
PRODUCT	product name		U	C	V	T
PROD_DIR	product root directory (usually defined relative to PROD_DIR_PREFIX, below)				V	
PROD_DIR_PREFIX	product root directory prefix (area where all or most product instances are maintained)	D				
QUALIFIERS	additional instance specification information often used to indicate compilation options used by developer Notes: appears immediately after a FLAVOR in these files, and is coupled with it to complete the instance identification (see 26.2.3 *Qualifiers: Use in Instance Matching*)		U	C	V	T
SETUPS_DIR	location of `setups.[c]sh` files and other **UPS** initialization files	D				
STATISTICS	flag to record statistics for specified products See 27.6.3 *STATISTICS* for usage information.	D			V	
TABLE_DIR Default: search path (see section 28.4 *Determination of ups Directory and Table File Locations*)	location of table file				V	
TABLE_FILE	name of table file (relative to TABLE_DIR)				V	
UNWIND_ARCHIVE_FILE	(a **UPD** keyword used only on distribution server configurations) absolute path to directory in which to unwind archive file (tar file) of product		U			

Keyword and Default Value (if any)	Description and Notes (if any)	D	U	C	V	T
UNWIND_PROD_DIR	(a **UPD** keyword) absolute path to directory where product gets unwound		U			
UNWIND_TABLE_DIR	(a **UPD** keyword) absolute path to directory where the table file gets unwound		U			
UNWIND_UPS_DIR	(a **UPD** keyword) absolute path to directory where the `ups` directory gets unwound		U			
UPD_USERCODE_DB	Database containing UPD_USERCODE_DIR (set internally)					
UPD_USERCODE_DIR	Directory where **UPD** configuration files are maintained	D				
UPS_ARCHIVE_FILE	(a **UPD** keyword used only on distribution server configurations) archive file (tar file) location that **UPD** specifies in **`ups declare -T ftp://host${UPS_ARCHIVE_FILE}`**		U			
UPS_DB_VERSION	**UPS** database version	D		C	V	T
UPS_DIR Default: `${UPS_PROD_DIR}/ups` if directory exists there	location of `ups` directory (if not absolute path, then taken relative to PROD_DIR, if specified)				V	
UPS_PROD_DIR	(a **UPD** keyword) product root directory that **UPD** specifies in the **`ups declare -r`** option; should be defined relative to PROD_DIR_PREFIX for portability		U			
UPS_TABLE_DIR	(a **UPD** keyword) table file directory that **UPD** specifies in the **`ups declare -M`** option **Normally this should not be set**! Since `UPS_TABLE_DIR` must be an absolute path, the declaration becomes non-portable if you set this location.		U			
UPS_THIS_DB	(a **UPD** keyword) the database into which **UPS** declares the product (i.e., the directory that **UPD** specifies in the **`ups declare -z`** option).		U			

Keyword and Default Value (if any)	Description and Notes (if any)	D	U	C	V	T
UPS_UPS_DIR	(a **UPD** keyword) `ups` directory that **UPD** specifies in the **`ups declare -U`** option, taken relative to `${UNWIND_PROD_DIR}` unless an absolute path is given; usually defined as `ups`.		U			
UPS_TABLE_FILE	(a **UPD** keyword) table file name that **UPD** specifies in the **`ups declare -m`** option		U			
USER	current username					T
VERSION	product version			C	V	T
_UPD_OVERLAY	main product name for overlaid product Note: This keyword is user-defined from **UPS**'s point of view. It is included here because it is configured and used by **UPD**. Its use with overlaid products is described in section 27.6.6 *_UPD_OVERLAY*.					T

27.5 Syntax for Assigning Keyword Values

- Any keyword value that has multiple values uses a colon (`:`) to separate the subvalues. The value (i.e., the list of subvalues) may be surrounded by double quotation marks (`"..."`). Blanks within the double-quoted value are ignored; they are neither required nor prohibited.

  For example, the following are all equivalent:

  ```
  QUALIFIERS = debug:optimize
  QUALIFIERS = "debug:optimize"
  QUALIFIERS = "  debug: optimize"
  ```

- Whitespace is ignored except within the keyword values for DESCRIPTION, DECLARER and MODIFIER
- Leading whitespace is ignored.

- There are no line continuation characters; the entire keyword definition or function must appear on a single line.
- The "at" character (@) is defined for use with the keywords COMPILE_FILE, PROD_DIR, UPS_DIR and TABLE_FILE. See section 27.6 *Usage Notes on Particular Keywords*.

27.6 Usage Notes on Particular Keywords

27.6.1 COMPILE_DIR, COMPILE_FILE and @COMPILE_FILE

COMPILE_DIR	the directory in which the compile file resides (see Chapter 37: *Use of Compile Scripts in Table Files*)
COMPILE_FILE	the name of the file containing precompiled functions
@COMPILE_FILE	the entire path to the file containing precompiled functions

27.6.2 PROD_DIR_PREFIX, PROD_DIR and @PROD_DIR

PROD_DIR_PREFIX	is generally set to the root of the path shared by all the products.
PROD_DIR	is the path that gets specified when the particular product instance is declared; it is usually (but not always) a relative path that gets tacked onto PROD_DIR_PREFIX.
@PROD_DIR	is a shorthand to request the entire path for the directory where the product is installed (usually equivalent to PROD_DIR_PREFIX/PROD_DIR).

If PROD_DIR_PREFIX is not defined on your system, then PROD_DIR should represent the entire path, in which case PROD_DIR and @PROD_DIR are identical.

Products installed prior to the upgrade to **UPS** v4 often reside in a different area than the newer products, and you may find that PROD_DIR_PREFIX is not set properly for them.

Compare these commands and their output:

```
% ups list -K PROD_DIR_PREFIX teledata
    "/afs/fnal.gov/ups/prd"
% ups list -K PROD_DIR teledata
    "teledata/v1_0/NULL"
% ups list -K @PROD_DIR teledata
    "/afs/fnal.gov/ups/prd/teledata/v1_0/NULL"
```

27.6.3 STATISTICS

The STATISTICS keyword is provided to allow recording of the following statistics on product usage and **UPS** database access:

- Userid of person executing **UPS/UPD** command
- Date and time

- Which command was executed (including options and arguments)
- Which product instance was selected by command

This keyword can appear in a product's version file and/or in the **UPS** database configuration file, thus providing a great deal of flexibility in choosing which products/instances to monitor.

Use in a Version File

When the STATISTICS keyword is present in a version file, it must be included with each specific instance which is to be monitored. If the STATISTICS keyword is located *before* any FLAVOR and/or QUALIFIERS keywords (these keywords separate out different instances), then it is ignored. In a version file, this keyword should have no value assigned.

Use in a Database Configuration File

When the STATISTICS keyword appears in the database configuration file, it needs a value. (If it has no value, it is ignored.) Its value is a colon-separated list of the products (name only) on which to record statistics (e.g., `STATISTICS = "tcl:tk:cern"`). The value `*` (asterisk) indicates that statistics are to be gathered on all products in the database.

Statistics Output

For a given product being monitored, statistics data for the product get recorded in a file whose name is the same as the product. If the product has dependencies, data also get recorded for them in their own product-specific files, and the data include the parent product name and version number. The data get recorded only when the **UPS/UPD** command in question has succeeded (i.e., when the temporary file has been created, but not yet sourced).

The statistics output files for all the monitored products and their dependencies reside in a special directory associated with the **UPS** database, namely `$PRODUCTS/.upsfiles/statistics`. This makes it easy to determine which products are being monitored, and only one directory needs to be made world-writable.

As an example of the statistics data that get recorded, let's look at the **tcl** product. It is a dependency of **tk**. Data that are recorded when an instance of **tcl** is accessed independently look like this:

```
"tcl" "v8_0" "IRIX" "" "" "user1" "2000-03-18 15.22.36 GMT" "setup"
```

Data that are recorded for **tcl** when an instance of **tk** is accessed look like this:

```
"tcl" "v8_0" "IRIX" "" "" "user1" "2000-03-18 15.22.36 GMT" "setupRequired tk v8_0"
```

27.6.4 TABLE_FILE and @TABLE_FILE

TABLE_FILE represents only the name of the table file, not its path. @TABLE_FILE is the entire path for the table file. Compare these commands and their output:

```
% ups list -Ktable_file teledata
```

```
"v1_0.table"
```

```
% ups list -K@table_file teledata
```

```
"/afs/fnal.gov/ups/db/teledata/v1_0.table"
```

See section 28.4 *Determination of ups Directory and Table File Locations* for information on how **UPS** determines the table file directory.

27.6.5 UPS_DIR and @UPS_DIR

UPS_DIR represents the location of the product's `ups` directory. If it is not an absolute path, then it is taken relative to @PROD_DIR (as shown in the example below). @UPS_DIR is the absolute path. Compare these commands and their output:

```
% ups list -K @PROD_DIR teledata
    "/afs/fnal.gov/ups/prd/teledata/v1_0/NULL"
% ups list -Kups_dir teledata
    "ups"
% ups list -K@ups_dir teledata
    "/afs/fnal.gov/ups/prd/teledata/v1_0/NULL/ups"
```

27.6.6 _UPD_OVERLAY

The _UPD_OVERLAY keyword defined in **UPD**[1] is provided for inclusion in the table file of each overlaid product. Overlaid products are introduced in section 1.3.7 *Product Overlays* and discussed again for developers in section 16.2.4 *Overlaid Products*. _UPD_OVERLAY takes as its value the main product name in double quotes. Its presence indicates that the product is an overlaid product maintained in the root directory of the main product listed as the keyword's value. For example, the table files for the products **cern_bin**, **cern_ups**, and **cern_lib** would contain the following keyword line:

```
_UPD_OVERLAY = "cern"
```

UPD would then use **cern** as the product name when determining the root directory.

1. **UPS** regards the _UPD_OVERLAY keyword as user-defined.

Chapter 28: Version Files

Version files are **UPS** database files that contain information specific to the local installation and declaration of the declared product instances. The contents of version files are described in this chapter.

28.1 About Version Files

The information in a version file includes (but is not limited to):

- when the instance was declared
- who declared the instance
- the product root directory of the instance
- the location of the `ups` directory
- the location of the table file for the instance

One version file must exist for each version of a product that is declared to the **UPS** database. For a particular version of a product, there is often a separate product instance installed for each flavor; and sometimes more than one per flavor if qualifiers are used. A new version file is created automatically by **UPS** when the first instance of a new version of a product is declared to the **UPS** database via the **`ups declare`** command. When a subsequent instance of the same version is declared, **UPS** automatically modifies the existing version file to include information for it. Multiple product instances are therefore often represented in a single version file.

The naming convention for version files is the version number followed by `.version`, e.g., `v19_34.version`. The version file must reside in the appropriate product-specific directory under the **UPS** database directory, `$PRODUCTS/<product>/<version>.version` (e.g., `$PRODUCTS/emacs/v19_34.version`).

The information in version files is stored in keyword definitions as described in 27.2 *Keywords: Information Storage Format*. The keywords get set according to the options specified on the **`ups declare`** command line.

28.2 Keywords used in Version Files

This is a subset of the list given in section 27.4 *List of Supported Keywords*.

Keyword and Default Value (if any)	Description and Notes (if any)
ARCHIVE_FILE	archive file name/location; used by **UPD**
AUTHORIZED_NODES Default: All nodes (*); taken from **UPS** database configuration file	authorized nodes
COMPILE_DIR	directory in which the compile file resides
COMPILE_FILE	the name of the file containing compiled functions (see Chapter 37: *Use of Compile Scripts in Table Files*)
DECLARED Default: current date and time	the date/time that the instance was declared to **UPS** or declared with a chain Note: often has multiple values, one for each declaration (e.g., for subsequent chain declarations)
DECLARER Default: current user	userid of user that performed the declaration Note: often has multiple values, one for each declaration (e.g., for subsequent chain declarations)
DESCRIPTION	product description
FILE	type of file (possible values: DBCONFIG, UPDCONFIG, CHAIN, VERSION, TABLE)
FLAVOR	product instance flavor Note: To easily accommodate flavor-neutral **`setup`** functions in a table file, FLAVOR can take the value ANY, but *only* in a table file.
MODIFIED Default: Current date/time	last time the associated instance was changed Note: often has multiple values, one for each declaration/modification (e.g., for subsequent chain declarations)
MODIFIER Default: Current user	userid of user that modified the instance Note: often has multiple values, one for each declaration/modification (e.g., for subsequent chain declarations)
ORIGIN	master source file; see option **`-D`** in Chapter 24: *Generic Command Option Descriptions*
PRODUCT	product name
PROD_DIR	product root directory (usually defined relative to PROD_DIR_PREFIX, below)

Keyword and Default Value (if any)	Description and Notes (if any)
QUALIFIERS	additional instance specification information often used to indicate compilation options used by developer Notes: appears immediately after a FLAVOR in these files, and is coupled with it to complete the instance identification (see 26.2.3 *Qualifiers: Use in Instance Matching*)
STATISTICS	flag to record statistics for specified products See section 11.8.3 *Collecting Statistics on Product Usage* for usage information.
TABLE_DIR Default: search path (see section 28.4 *Determination of ups Directory and Table File Locations*)	location of table file
TABLE_FILE	name of table file (relative to TABLE_DIR)
UPS_DB_VERSION	**UPS** database version
UPS_DIR Default: `${UPS_PROD_DIR}/ups` if directory exists there	location of `ups` directory (if not absolute path, then taken relative to PROD_DIR, if specified)
VERSION	product version

28.3 Version File Examples

28.3.1 Sample Version File for exmh v1_6_6

Let's declare a new version of **exmh** via the command:

```
% ups declare -r /afs/fnal.gov/products/UNIX/exmh/v1_6_6 \
  -m exmh.table exmh v1_6_6
```

This example assumes the `ups` directory resides in its default location (directly under product root directory), the table file resides in a default location (see section 28.4 *Determination of ups Directory and Table File Locations*) and we are using $PRODUCTS to determine the database (**`-U <upsDir>`**, **`-M <tableFileDir>`** and **`-z <databaseList>`** are unspecified).

Given a machine of flavor SunOS+5, this creates the following version file, named `v1_6_6.version`:

```
FILE = version
PRODUCT = exmh
VERSION = v1_6_6

#***********************************************
#
FLAVOR = SunOS+5
QUALIFIERS = ""
  DECLARED = 1998-03-30 21.06.59 GMT
  DECLARER = stolz
  MODIFIED = 1998-03-30 21.06.59 GMT
  MODIFIER = stolz
  PROD_DIR = /afs/fnal.gov/products/UNIX/exmh/v1_6_6
  UPS_DIR = ups
  TABLE_FILE = exmh.table
```

28.3.2 Sample version file for foo v2_0

Version files can contain information for multiple instances of a single version of a product. Here is an example for a fictional product **foo** v2_0. The file below would have been created and modified by the series of commands:

```
% ups declare foo v2_0 -m v2_0.table -f IRIX -q superoptimize \
  -r /usr/prod/IRIX/foo/v2_0s

% ups declare foo v2_0 -m v2_0.table -f OSF1 \
  -r /usr/prod/OSF1/foo/v2_0
```

```
FILE = version
PRODUCT = foo
VERSION = v2_0

#***********************************************
#
FLAVOR = IRIX
QUALIFIERS = "superoptimize"
  DECLARER = aheavey
  DECLARED = 1998-04-15 16.37.58 GMT
  MODIFIER = aheavey
  MODIFIED = 1998-04-15 16.37.58 GMT
  PROD_DIR = /usr/prod/IRIX/foo/v2_0s
  UPS_DIR = ups
  TABLE_FILE = v2_0.table

 #---------------------------------------
 #
 FLAVOR = OSF1
 QUALIFIERS = ""
  DECLARER = aheavey
  DECLARED = 1998-04-15 16.39.58 GMT
  MODIFIER = aheavey
  MODIFIED = 1998-04-15 16.39.58 GMT
   PROD_DIR = /usr/prod/OSF1/foo/v2_0
   UPS_DIR  = ups
   TABLE_FILE = v2_0.table
```

28.4 Determination of ups Directory and Table File Locations

In a version file, the TABLE_DIR and UPS_DIR keywords can each be specified as an absolute or a relative path. When either is specified as a *relative* path, it is taken as relative to PRODUCT_DIR_PREFIX/PRODUCT_DIR[1].

The table file name and directory can be specified in several ways, depending on how their corresponding keywords have been defined. **UPS** uses the following algorithm to determine the table file location:

If TABLE_FILE is specified as an absolute path, then:

- The location is TABLE_FILE.

If TABLE_FILE is specified as a relative path, or simply as the filename, then:

- If TABLE_DIR is specified, the location is TABLE_DIR/TABLE_FILE.
- If TABLE_DIR is not specified, and UPS_DIR is specified, then two locations are searched: first the product subdirectory in the database (e.g., `$PRODUCTS/<product>`), and second UPS_DIR.
- If neither TABLE_DIR nor UPS_DIR is specified at all, **UPS** will search for TABLE_FILE under the product subdirectory in the database only.

1. Be aware that PROD_DIR_PREFIX may not be defined; if not, PROD_DIR should be an absolute path.

Chapter 29: Chain Files

UPS/UPD supports *chains* to product versions, and chain information is maintained in chain files. In this chapter we describe chain files and how they interact with version files.

29.1 About Chain Files

Chains for a product are maintained in chain files which reside in the product-specific directory under the **UPS** database directory. There is one chain file for each chain name, and it is named according to the chain name, with a suffix of `.chain`, e.g., `current.chain`. A chain file is automatically created by **UPS** the first time an instance of a product is declared with some chain. When any other instances of the same product (regardless of version) get declared with the same chain, one of two things happens:

- a new entry is created in the same chain file, or
- if an entry with the same flavor and qualifiers already exists, the pre-existing entry gets unchained and the new one is chained in its place.

Chain files get created and modified via the **`ups declare <chainFlag>`** command. A chain file's contents are simply a formatted list of the product instances that were declared with that chain, where each product instance is specified via a set of keywords. When a chain is used in a **UPS/UPD** command, **UPS** looks in the corresponding chain file to match the instance and thus locate the appropriate version file. As discussed in section 26.2 *Instance Matching within Selected Database*, the version file entry locates the product root directory and table file to retrieve the instance.

In **UPS/UPD** commands, the command line option associated with a particular chain can be used in specifying the product instance to match. Using chains is optional, but recommended. Both chained and unchained instances of a product may be declared to **UPS**; the user can still retrieve any instance, chained or not, by specifying its version number.

29.2 Keywords Used in Chain Files

This is a subset of the list given in section 27.4 *List of Supported Keywords*.

Keyword and Default Value (if any)	Description and Notes (if any)
CHAIN	chain name
DECLARED Default: current date and time	the date/time that the instance was declared to **UPS** or declared with a chain Note: often has multiple values, one for each declaration (e.g., for subsequent chain declarations)
DECLARER Default: current user	userid of user that performed the declaration Note: often has multiple values, one for each declaration (e.g., for subsequent chain declarations)
DESCRIPTION	product description
FILE	type of file (possible values: DBCONFIG, UPDCONFIG, CHAIN, VERSION, TABLE)
FLAVOR	product instance flavor Note: To easily accommodate flavor-neutral **`setup`** functions in a table file, FLAVOR can take the value ANY, but *only* in a table file.
MODIFIED Default: Current date/time	last time the associated instance was changed Note: often has multiple values, one for each declaration/modification (e.g., for subsequent chain declarations)
MODIFIER Default: Current user	userid of user that modified the instance Note: often has multiple values, one for each declaration/modification (e.g., for subsequent chain declarations)
PRODUCT	product name
QUALIFIERS	additional instance specification information often used to indicate compilation options used by developer Notes: appears immediately after a FLAVOR in these files, and is coupled with it to complete the instance identification (see 26.2.3 *Qualifiers: Use in Instance Matching*)
UPS_DB_VERSION	**UPS** database version
VERSION	product version

29.3 Chain File Examples

29.3.1 Sample chain file for exmh v1_6_6

This file points to the instance used in the version file of section 28.3.1 *Sample Version File for exmh v1_6_6*. The file `$PRODUCTS/exmh/current.chain` contains the text:

```
FILE = chain
PRODUCT = exmh
CHAIN = current

#*********************************************
#
FLAVOR = SunOS+5
QUALIFIERS = ""
  VERSION = v1_6_6
  DECLARED = 1998-03-30 21.06.59 GMT
  DECLARER = stolz
  MODIFIED = 1998-03-30 21.06.59 GMT
  MODIFIER = stolz
```

If the given instance hadn't been initially declared as current (as in the command in section 28.3.1), then to create this chain file you would need to declare the instance current, e.g.,:

```
% ups declare -c exmh v1_6_6
```

29.3.2 Sample chain file for foo v2_0

This example illustrates the use of qualifiers. It points to both of the instances in the version file for **foo** in section 28.3.2 *Sample version file for foo v2_0*. That version file will also get modified when these chains are declared. The DECLARER, DECLARED, MODIFIER and MODIFIED fields will include information for the chain declarations.

Making the "current" Chain Declarations

In order for this chain file to have the contents shown below, the following two commands need to be issued:

```
% ups declare -cq superoptimize -f IRIX foo v2_0
```

```
% ups declare -cf OSF1 foo v2_0
```

The file `$PRODUCTS/foo/current.chain` contains the text:

```
FILE = CHAIN
PRODUCT = foo
CHAIN = CURRENT
#
#--------------------------------------
#
FLAVOR = IRIX
QUALIFIERS = "superoptimize"
  VERSION = v2_0
  DECLARER = aheavey
  DECLARED = 1998-04-15 16.37.58 GMT
  MODIFIED = 1998-05-19 21.06.59 GMT
  MODIFIER = aheavey
```

```
FLAVOR = OSF1
QUALIFIERS = ""
  VERSION = V2_0
  DECLARER = aheavey
  DECLARED = 1998-04-15 16.39.58 GMT
  MODIFIED = 1998-05-24 21.06.59 GMT
  MODIFIER = aheavey
```

Sequence of Events at Setup Time

For this example in the IRIX case, the sequence of events upon issuing the command:

```
% setup -q superoptimize foo
```

would be as follows:

1) match the FLAVOR (IRIX) and the QUALIFIERS (superoptimize) in this chain file

2) find the version (v2_0) and open the corresponding version file (`v2_0.version`)

3) locate the table file (`$FOO_DIR/ups/v2_0.table`) and open it

4) find the ACTION=SETUP line in the table file and execute the listed functions (if no ACTION=SETUP line is present, **UPS** executes the default **`setup`** functions)

Chapter 30: The UPS Configuration File

A **UPS** database can be configured and customized using the file `dbconfig`, described in this chapter. This file is usually maintained in the location `$PRODUCTS/.upsfiles/dbconfig`. It is used to define keywords which control quantities such as:

- which nodes can access products maintained in the database
- the directory under which products are installed
- which products will have usage statistics collected
- the directories for product **man** pages and **Info** files
- the directory containing the **UPS** initialization files
- the directory containing the **UPD** configuration file
- the **UPS** database version

A template `dbconfig` file is available in `$UPS_DIR/ups/dbconfig.template`.

30.1 dbconfig File Organization

The `dbconfig` file consists of keyword definitions. It always has as its first line:

```
File = dbconfig
```

to identify itself to the system. The additional keywords can be in any order.

30.2 Keywords Used in dbconfig

Keyword	Description and Notes
AUTHORIZED_NODES	nodes authorized to access the database (set value to "*" to allow all nodes access; for a list of nodes, separate nodes with colons)
CATMAN_TARGET_DIR	directory into which catman files are to be copied
FILE	type of file (must be set to DBCONFIG)

Keyword	Description and Notes
HTML_TARGET_DIR	directory into which html files are to be copied *Not yet supported.*
INFO_TARGET_DIR	directory into which **Info** files are to be copied
MAN_TARGET_DIR	directory into which formatted man pages are to be copied
NEWS_TARGET_DIR	directory into which news files are to be copied (for posting to a newsgroup) *Not yet supported.*
PROD_DIR_PREFIX	product root directory prefix (directory under which all or most product instances are maintained); must be an absolute path
SETUPS_DIR	location of `setups.[c]sh` files and other **UPS** initialization files (note that "courtesy links" in `/usr/local/etc` should be created to point to this directory; see section 1.7.1 *Initializing the UPS Environment*)
STATISTICS	flag to record statistics for specified products (see section 27.6.3 *STATISTICS* for usage information)
UPD_USERCODE_DIR	directory where **UPD** configuration file is maintained (usually `$PRODUCTS/.updfiles/`)
UPS_DB_VERSION	**UPS** database version

30.3 Sample dbconfig File

```
FILE = DBCONFIG

# all nodes can read/use the products declared in this db
AUTHORIZED_NODES = *

# all product roots are under /fnal/ups/prd
PROD_DIR_PREFIX = /fnal/ups/prd

# keep statistics about the following products:
#   (uncomment to get stats!)
# STATISTICS = ups:upd:perl

# manpages, info files, get copied here:
MAN_TARGET_DIR = /fnal/ups/man
CATMAN_TARGET_DIR = /fnal/ups/catman
# INFO_TARGET_DIR = /fnal/ups/Info

#  automatic html and news processing not yet supported
#  HTML_TARGET_DIR = /dev/null
#  NEWS_TARGET_DIR = /dev/null
```

```
#  setups.[c]sh scripts are copied here
#  ('courtesy links' in /usr/local/etc should point here):
SETUPS_DIR = /fnal/ups/etc

#   upd configuration for this db are here:
UPD_USERCODE_DIR = /fnal/ups/db/.updfiles
```

Chapter 31: The UPD Configuration File

UPD can be configured and customized on your system using the file `updconfig`, described in this chapter. This file is usually maintained in the location `$PRODUCTS/.updfiles/updconfig`. (Its location is commonly referred to as `$UPD_USERCODE_DIR/updconfig`, where $UPD_USERCODE_DIR gets defined in the `dbconfig` file of the **UPS** database in which **UPD** is declared.) By providing default values for several variables (mostly product file and directory locations), the `updconfig` file controls where **UPD** installs products and miscellaneous product-related files.[1] It can also be used to define supplementary actions for **UPD** to perform when installing or updating products. Use of the `updconfig` file greatly reduces the amount of information the installer/maintainer needs to provide to the system for each **UPD** operation. A template `updconfig` file is available in `$UPD_DIR/ups/updconfig.template`.

Locations defined in a `updconfig` file may be overridden by specifying corresponding options on the **UPD** command line.

31.1 updconfig File Organization

The `updconfig` file always has as its first line:

```
File = updconfig
```

to identify itself to the system. The remainder of the file consists of one or more stanzas. Each stanza:

- identifies certain product instances, products or groups of products
- specifies a database on the local system in which to declare a matched product
- specifies locations on the local system in which **UPD** is to put a matched product and its related files
- (optionally) lists actions for **UPS/UPD** to perform either just before or just after declaring the matched product

A `updconfig` file stanza is of the form:

```
GROUP:
...
COMMON:
...
END:
```

1. In **UPS/UPD** versions prior to and including v4_4a, the file `${UPD_USERCODE_DIR}/updusr.pm` could be used to override the default behavior of **UPD**. This file is now obsolete and can be removed.

The `GROUP:` section of the stanza contains the product matching information. The `COMMON:` section contains the locations and actions for the matched product and its associated files. `END:` is used to end the stanza.

31.2 Product Instance Identification and Matching

There are several identifiers which can be used to specify a product instance match. When multiple values are listed for an identifier, the logical "or" of those values is used. Identifiers which are omitted default to `ANY`, which means they match any product instance. The following identifiers are supported:

`product`	product name
`flavor`	flavor string
`qualifiers`	qualifier string
`options`	option (anything specified via the **`-O`** (uppercase **`-o`**) option in the **UPD** command
`dist_database`	database path on the distribution server
`dist_node`	node name of distribution server

As an example, the following stanza identifies the product **exmh**, for either the flavor SunOS+5.5 or IRIX+6.3 and any qualifiers (omitted, therefore set to `ANY` by default) on *fnkits.fnal.gov*:

```
GROUP:
  product       = exmh
  flavor        = SunOS+5.5, IRIX+6.3
  dist_node     = fnkits.fnal.gov
COMMON:
  ...
END:
```

In the current implementation, the first stanza to match a given product instance is the one that gets used; **UPD** does not continue searching in the file for a "better" match.[1]

1. In the future, we hope to have a more flexible configuration file parser, more in line with **UPS** table files. We plan to make those rules upward-compatible relative to the current ones.

31.3 Defining Locations for Product Files

Within each stanza, file and directory locations for installing matched product instances and their associated files must be defined. These locations should be defined in terms of **UPS/UPD** read-only variables.

31.3.1 Required Locations

All the locations/keywords listed in the table below are required (the last two for distribution nodes only).

Note: `UPS_THIS_DB`, listed first, is used by **UPD** to determine the database in which to look for `PROD_DIR_PREFIX` (set in `dbconfig`, see Chapter 30: *The UPS Configuration File*). Several of the keywords that follow may be defined relative to its corresponding read-only variable `${PROD_DIR_PREFIX}`.

`UPS_THIS_DB`	the database into which **UPS** declares the product (i.e., the directory that **UPD** specifies in the **`ups declare -z`** option). Recommendation: Set it to `${UPD_USERCODE_DB}`, which is the database in which the `updconfig` file was found.
`UPS_PROD_DIR`	product root directory that **UPD** specifies in the **`ups declare -r`** option; should be defined relative to `${PROD_DIR_PREFIX}` for portability
`UNWIND_PROD_DIR`	absolute path to directory where product gets unwound In most cases, it's `${PROD_DIR_PREFIX}/${UPS_PROD_DIR}`, however in AFS and some NFS mounting configurations, products are often unwound and declared in different locations (see section 8.3 *Installing Products into AFS Space*).
`UPS_UPS_DIR`	`ups` directory that **UPD** specifies in the **`ups declare -U`** option, taken relative to `${UNWIND_PROD_DIR}` unless an absolute path is given; usually defined as `ups.`
`UNWIND_UPS_DIR`	absolute path to directory where the `ups` directory gets unwound; usually defined as `${UNWIND_PROD_DIR}/${UPS_UPS_DIR}` or `${UNWIND_PROD_DIR}/ups`.
`UPS_TABLE_DIR`	table file directory that **UPD** specifies in the **`ups declare -M`** option **Normally this should not be set**! In some cases you may need to put the table file somewhere other than where **UPS** will automatically look (namely `$PROD-UCTS/${UPS_PROD_NAME}` and `${UPS_UPS_DIR}`); however since UPS_TABLE_DIR must be an absolute path, the declaration becomes non-portable if you set this location.

UNWIND_TABLE_DIR	absolute path to directory where the table file gets unwound Suggestion: To maintain one table file for all flavors of a product, put it in the database; i.e., set this to `${UPS_THIS_DB}/${UPS_PROD_NAME}`. To maintain each table file under `$<PRODUCT>_DIR/ups`, set it to `${UPS_UPS_DIR}`.
UPS_TABLE_FILE	table file name that **UPD** specifies in the **`ups declare -m`** option Depending on where you maintain table files, choose a naming convention that identifies each file adequately. For example, if you maintain all product table files in one location, the filename should include the product name and version (e.g., `${UPS_PROD_NAME}_${UPS_PROD_VERSION}.table`); if each is kept under its product root directory, the product name is not necessary (e.g., `${UPS_PROD_VERSION}.table`).
UNWIND_ARCHIVE_FILE	absolute path to directory in which to unwind archive file (tar file) of product Used only on distribution server configurations.
UPS_ARCHIVE_FILE	archive file (tar file) location that **UPD** specifies in **`ups declare -T ftp://host${UPS_ARCHIVE_FILE}`** Used only on distribution server configurations.

31.3.2 Read-Only Variables Usable in Location Definitions

The following predefined **UPS/UPD** read-only variables can be used in the definition of locations described above. These variables get their values from the command line and/or the dependency list.

Do not try to redefine these variables. When you use these variables, always enclose them in curly brackets ({}) as shown in the list.

${UPS_USERCODE_DB}	database containing **UPD** configuration
${UPS_USERCODE_DIR}	directory containing **UPD** configuration
${UPS_PROD_NAME}	name of product
${UPS_PROD_FLAVOR}	flavor of product
${UPS_PROD_QUALIFIERS}	qualifiers of product
${UPS_BASE_FLAVOR}	flavor trimmed as in **`ups flavor -1`**
${DASH_PROD_FLAVOR}	flavor with non-word characters replaced by dashes; e.g., if {UPS_PROD_FLAVOR} is set to `SunOS+5`, then {DASH_PROD_FLAVOR} has the value `SunOS-5`. This is used to avoid problems with unusual symbols in file and directory names.

${DASH_PROD_QUALIFIERS}	qualifier list with non-word characters replaced by dashes; e.g., if {UPS_PROD_QUALIFIERS} is `qual1+qual2`, then {DASH_PROD_QUALIFIERS} is `qual1-qual2`. This is used to avoid problems with unusual symbols in file and directory names.
${SUFFIX}	suffix of archive file; e.g., `tar`, `zip`, etc. Used only on distribution server configurations.
${PROD_DIR_PREFIX}	`PROD_DIR_PREFIX` of database, defined in **UPS** configuration file `dbconfig` (see Chapter 30)

31.3.3 Sample Location Definitions

The following partial stanza, taken from the example in section 31.5.1, shows several location specifications:

```
COMMON:
     UPS_THIS_DB  = "${UPD_USERCODE_DB}"
     UPS_PROD_DIR = "${UPS_PROD_NAME}/${UPS_PROD_VERSION}/${DASH_PROD_FLAVOR}
                     ${DASH_PROD_QUALIFIERS}"  (this must be all on one line in the real file)
  UNWIND_PROD_DIR = "${PROD_DIR_PREFIX}/${UPS_PROD_DIR}"
      UPS_UPS_DIR = "ups"
   UNWIND_UPS_DIR = "${UNWIND_PROD_DIR}/${UPS_UPS_DIR}"

    #    Default (do not actually set UPS_TABLE_DIR):
    #   UPS_TABLE_DIR = "${UPS_THIS_DB}/${UPS_PROD_NAME}"
     UNWIND_TABLE_DIR = "${UPS_TABLE_DIR}"
       UPS_TABLE_FILE = "${UPS_PROD_VERSION}.table"
...
    END:
```

31.4 Pre- and Postdeclare Actions

An *action* is a construction that identifies a **UPS** or user-defined operation via the ACTION keyword, and lists functions to perform, in addition to any internal processes, when the operation is executed. An action stanza has the format:

```
ACTION=<VALUE>
   <function_1>([<argument_1>] [, <argument_2>] ...)
   <function_2>([<argument_1>] [, <argument_2>] ...)
   ...
```

The `updconfig` file uses actions to define the steps **UPD** is to perform during an installation/update of the matched product instance(s). The ACTION keyword values indicate when to perform the steps (before or after issuing the **`ups declare`** command), and the steps themselves are listed as functions under the ACTION line. In a `updconfig` file, actions can be listed anywhere in the `COMMON:` part of a stanza.

31.4.1 ACTION Keyword Values

Currently two action keyword values are supported for use in `updconfig`:

`predeclare`	Perform listed functions after product files have been unwound, but before the product has been declared
`postdeclare`	Perform listed functions after product has been declared

Functions are then listed after the ACTION keyword line, using the following syntax:

```
Action = predeclare
    function(arg,arg,...)
    function(arg,arg,...)
```

31.4.2 The execute Function

Currently, only the **`execute`** function is supported for use in `updconfig`:

`execute("<command>", <UPS_ENV_FLAG>, [, <VARIABLE>])`

It executes a shell-independent command and (optionally) assigns the output to an environment variable, **`<VARIABLE>`**. It takes a required parameter (**`UPS_ENV_FLAG`**) which indicates whether to define **UPS** local variables. This parameter can take the following values:

UPS_ENV	define all local **UPS** environment variables before sourcing (the script or command relies on these being defined)
NO_UPS_ENV	do not define the local **UPS** environment variables (the script or command doesn't use them)

If the optional third argument, **`<VARIABLE>`**, is not specified, then the specified command is executed but the output from that command is not saved. This command does not have to be shell-independent.

For example, say that you want to make group-writable the directory in which a product has been unwound before it is declared. You would include the action:

```
ACTION = PREDECLARE
    execute ("chmod -R g+w ${UNWIND_PROD_DIR}", NO_UPS_ENV )
```

31.5 Examples

31.5.1 Generic Template updconfig File

This example is taken from the template, `$UPD_DIR/ups/updconfig.template` (comments are included in the actual template file). The template is designed to be usable *as is*, if:

- you have only one **UPS** database
- you want your product root hierarchy to be:

  `${PROD_DIR_PREFIX}/<product>/<version>/<flavor><qualifiers>`
- you want your table files to reside in the **UPS** database as:

  `${UPS_THIS_DB}/<product>/<version>.table`

If your requirements are different, this file is still useful as a starting point from which to make modifications.

```
File = updconfig
#
GROUP:
  product       = ANY
  flavor        = ANY
  qualifiers    = ANY
  options       = ANY
  dist_database = ANY
  dist_node     = ANY

COMMON:
     UPS_THIS_DB  = "${UPD_USERCODE_DB}"
     UPS_PROD_DIR =  "${UPS_PROD_NAME}/${UPS_PROD_VERSION}/${DASH_PROD_FLAVOR}
                      ${DASH_PROD_QUALIFIERS}" (no line break in real file)
  UNWIND_PROD_DIR = "${PROD_DIR_PREFIX}/${UPS_PROD_DIR}"
      UPS_UPS_DIR = "ups"
   UNWIND_UPS_DIR = "${UNWIND_PROD_DIR}/${UPS_UPS_DIR}"

#    Default (do not actually set UPS_TABLE_DIR):
#   UPS_TABLE_DIR = "${UPS_THIS_DB}/${UPS_PROD_NAME}"
 UNWIND_TABLE_DIR = "${UPS_TABLE_DIR}"
   UPS_TABLE_FILE = "${UPS_PROD_VERSION}.table"
#
#    Possible alternative, where the table files live
#    within the product's ups directory.  Note,
#    in this case you ALSO should not set UPS_TABLE_DIR.
#
##  UPS_TABLE_DIR = "${UNWIND_UPS_DIR}"
#UNWIND_TABLE_DIR = "${UPS_TABLE_DIR}"
# UPS_TABLE_FILE  = "${UPS_PROD_NAME}.table"
#
#  ACTION = PREDECLARE
#           add functions
#  ACTION = POSTDECLARE
#           add functions
END:
```

31.5.2 Distribution from the fnkits Node Only

As a second example, we show the `GROUP:` portion of a file that specifies a particular distribution host. Aside from the `dist_node` entry, the stanza is identical to that of the template `updconfig` file, and therefore applies to any products coming from the specified host, *fnkits.fnal.gov*. *fnkits* is the central Computing Division product server, and there are several names for it. All the names are all listed and delimited by colons.

```
File = updconfig

GROUP:

  product       = ANY
  flavor        = ANY
  qualifiers    = ANY
  options       = ANY
  dist_database = ANY
  dist_node     = fnkits:fnkits.fnal.gov:kits:kits.fnal.gov:upd:upd.fnal.gov

COMMON:
...
END:
```

This **UPD** configuration file could be expanded to include additional stanzas to accommodate products from other distribution nodes.

31.5.3 Customized Treatment of ups Directory and Table Files

In this example, the distribution node again has changed relative to the template, and this time there are also changes in the `COMMON:` section. The distribution node is `e007.dist.xyz.edu`. UPS_PROD_DIR is no longer defined relative to PROD_DIR_PREFIX, but is now placed under the `/e007/base_code` directory. All the table files are placed in a single directory (`/e007/table_files`), therefore the table file names must include the product name in order to be identifiable. Here they will be named `<product>_<version>.table` (defined using the corresponding variables as `${UPS_PROD_NAME}_${UPS_PROD_VERSION}.table`).

```
File = updconfig

GROUP:
  product       = ANY
  flavor        = ANY
  qualifiers    = ANY
  options       = ANY
  dist_database = ANY
  dist_node     = e007_dist.xyz.edu

COMMON:
      UPS_THIS_DB      = "${UPD_USERCODE_DB}"
     UPS_PROD_DIR     = "/e007/base_code/${UPS_PROD_NAME}/${UPS_PROD_VERSION}/
                        ${UPS_PROD_FLAVOR}${UPS_PROD_QUALIFIERS}" (no line break in real file)
     UNWIND_PROD_DIR  = "${PROD_DIR_PREFIX}/${UPS_PROD_DIR}"
     UPS_UPS_DIR      = "ups"
     UNWIND_UPS_DIR   = "${UNWIND_PROD_DIR}/${UPS_UPS_DIR}"
     UPS_TABLE_DIR    = "/e007/table_files"
     UNWIND_TABLE_DIR = "${UPS_TABLE_DIR}"
     UPS_TABLE_FILE   = "${UPS_PROD_NAME}_${UPS_PROD_VERSION}.table"
END:
```

31.5.4 Implementing Multiple Configurations

Here is an example that shows how to configure the file if more than one database and distribution node are used. The first section instructs **UPD** where to unwind products that are distributed from *fnkits* and how to declare them. The second section, with different naming conventions and file hierarchies, instructs **UPD** where to unwind and how to declare products obtained from the CDF distribution node *cdf-kits.fnal.gov.*

```
File = updconfig

GROUP:

  product       = ANY
  flavor        = ANY
  qualifiers    = ANY
  options       = ANY
  dist_database = ANY
  dist_node     = fnkits:fnkits.fnal.gov:kits:kits.fnal.gov:upd:upd.fnal.gov

COMMON:

     UPS_THIS_DB      = "${UPD_USERCODE_DB}"
     UPS_PROD_DIR     = "${UPS_PROD_NAME}/${UPS_PROD_VERSION}/${UPS_PROD_FLAVOR}
                         ${UPS_PROD_QUALIFIERS}"      (no line break in real file)
     UNWIND_PROD_DIR  = "${PROD_DIR_PREFIX}/${UPS_PROD_DIR}"
     UPS_UPS_DIR      = "ups"
     UNWIND_UPS_DIR   = "${UNWIND_PROD_DIR}/${UPS_UPS_DIR}"
###  UPS_TABLE_DIR    = "${UPS_THIS_DB}/${UPS_PROD_NAME}"
     UNWIND_TABLE_DIR = "${UPS_TABLE_DIR}"
     UPS_TABLE_FILE   = "${UPS_PROD_VERSION}.table"

END:

GROUP:

  product       = ANY
  flavor        = ANY
  qualifiers    = ANY
  options       = ANY
  dist_database = ANY
  dist_node     = cdf-kits.fnal.gov

COMMON:

     UPS_THIS_DB      = "~cdfsoft/declare"
     UPS_PROD_DIR     = "${UPS_PROD_NAME}/${UPS_PROD_VERSION}/${UPS_PROD_FLAVOR}
                         ${UPS_PROD_QUALIFIERS}"   (no line break in real file)
     UNWIND_PROD_DIR  = "/cdf/products/${UPS_PROD_NAME}/${UPS_PROD_VERSION}"
     UPS_UPS_DIR      = "ups"
     UNWIND_UPS_DIR   = "${UNWIND_PROD_DIR}/${UPS_UPS_DIR}"
###  UPS_TABLE_DIR    = "${UPS_THIS_DB}/${UPS_PROD_NAME}"
     UNWIND_TABLE_DIR = "${UPS_TABLE_DIR}"
     UPS_TABLE_FILE   = "${UPS_PROD_VERSION}.table"

END:
```

31.5.5 Sample Configuration for AFS Space Using ACTIONS

In AFS space, you may need to release the read-write volume before you can declare a product, as discussed in section 8.3 *Installing Products into AFS Space*. For this you would use a PREDECLARE action. You may also need to release the read-write **UPS** database after the product is declared, which can be done in a POSTDECLARE action. These actions are shown in this example.

```
File = updconfig

 GROUP:
  product       = ANY
  flavor        = ANY
  qualifiers    = ANY
  options       = ANY
  dist_database = ANY
  dist_node     = ANY

COMMON:
     UPS_THIS_DB = "/afs/.fnal.gov/ups/db"
     UPS_PROD_DIR = "${UPS_PROD_NAME}/${UPS_PROD_VERSION}/${UPS_PROD_FLAVOR}
                     ${UPS_PROD_QUALIFIERS}"  (no line break in real file)
  UNWIND_PROD_DIR = "/afs/.fnal.gov/ups/${UPS_PROD_DIR}"
      UPS_UPS_DIR = "ups"
   UNWIND_UPS_DIR = "${UNWIND_PROD_DIR}/${UPS_UPS_DIR}"
 UNWIND_TABLE_DIR = "${UPS_THIS_DB}/${UPS_PROD_NAME}"
   UPS_TABLE_FILE = "${UPS_PROD_VERSION}.table"

  ACTION = PREDECLARE
    Execute("/usr/local/bin/upd_volrelease ${UNWIND_PROD_DIR}", NO_UPS_ENV)
  ACTION = POSTDECLARE
    Execute("/usr/local/bin/upd_volrelease ${UPS_THIS_DB}", NO_UPS_ENV)

END:
```

31.5.6 Distribution Node Configuration

For this example, we present an abridged version of the `updconfig` file used on *fnkits*. The real one is fairly long and repetitive (will be shortened mid-2000), and is described in Chapter 21: *Configuration of the fnkits Product Distribution Node*

fnkits has a local database containing products destined for use on that node in addition to the `KITS` distribution database. On *fnkits*, one `updconfig` is used for both databases (this too will change mid-2000). It resides under the local database, and ${UPD_USERCODE_DIR} is defined accordingly in the `dbconfig` files for both databases.

The `updconfig` file on *fnkits* includes several stanzas, each of which pertains to a category of product. The product-matching criterion for each stanza is an `options=<option>` line which indicates the category[1]. This example shows only the stanzas used for ordinary distributed products (the default) and locally installed products. The default stanza is identified by the absence of an option, and the local stanza is identified by the option `local`.

The default stanza includes a PREDECLARE and a POSTDECLARE action. The PREDECLARE action contains a set of **`execute`** statements to **`chmod/chgrp`** the files to the right group id and permissions, and another set to symlink files under `/ftp/KITS` to provide the old-style (**UPS**/**UPD** v3) `KITS` hierarchy[2] of `KITS/Flavor/product/version`. The POSTDECLARE action makes a convenience tar file of the `ups` directory for users downloading via **FTP**. The stanza for `local` products contains no actions.

- Many of the location definitions and functions are quite long, and are shown here on multiple lines for readability.
- In the real file, each definition or function must be contained on a single line.

```
File=updconfig
#
group:
    # normal, ordinary products added to kits
   common:
    # actual locations of things
    UPS_THIS_DB = "/ftp/upsdb"
    UNWIND_PROD_DIR="/ftp/products/${UPS_PROD_NAME}/${UPS_PROD_VERSION}/
      ${UPS_PROD_FLAVOR}/${UPS_PROD_NAME}_${UPS_PROD_VERSION}_
      ${UPS_PROD_FLAVOR}${UPS_PROD_QUALIFIERS}"
    UNWIND_UPS_DIR = "${UNWIND_PROD_DIR}/ups"
    UNWIND_TABLE_DIR = "/ftp/products/${UPS_PROD_NAME}/${UPS_PROD_VERSION}/
      ${UPS_PROD_FLAVOR}"
    UNWIND_ARCHIVE_FILE = "${UNWIND_PROD_DIR}.${SUFFIX}"
    #
    # declared values of things
    UPS_TABLE_FILE = "${UPS_PROD_NAME}_${UPS_PROD_VERSION}_${UPS_PROD_FLAVOR}
       ${UPS_PROD_QUALIFIERS}.table"
    UPS_TABLE_DIR = "${UNWIND_TABLE_DIR}"
    UPS_PROD_DIR = "${UNWIND_PROD_DIR}"
    UPS_UPS_DIR = "ups"
    UPS_ARCHIVE_FILE = "${UNWIND_ARCHIVE_FILE}"

    action = predeclare
        #
        # fix group permissions
        Execute("chgrp upd ${UNWIND_TABLE_DIR}/*", NO_UPS_ENV)
        Execute("chmod o-rwx ${UNWIND_TABLE_DIR}/*", NO_UPS_ENV)
        Execute("chmod a+r ${UNWIND_TABLE_DIR}/*.table", NO_UPS_ENV)
       #
       # make old-KITS compatable hierarchy files

        Execute("test -d  /ftp/KITS/${UPS_BASE_FLAVOR}/${UPS_PROD_NAME}/
             ${UPS_PROD_VERSION} || mkdir -p /ftp/KITS/${UPS_BASE_FLAVOR}/
             ${UPS_PROD_NAME}/${UPS_PROD_VERSION}", NO_UPS_ENV)
```

---

1. For this type of configuration, unless some automatic implementation of option-matching is implemented (as is the case on *fnkits*), a product provider would need to include the appropriate option as **`upd addproduct -O <option>`** when adding the product to the distribution node, in order to invoke the right stanza. The option `local` is an exception: the person installing a product for local use on the distribution node would need to use **`upd install`** with the **`-O local`** option.
2. Currently nothing prunes old links or files from this hierarchy.

```
       Execute("cd /ftp/KITS/${UPS_BASE_FLAVOR}/${UPS_PROD_NAME}/${UPS_PROD_VERSION};
          rm -f ${UPS_PROD_NAME}_${UPS_PROD_VERSION}_
          ${UPS_PROD_FLAVOR}${UPS_PROD_QUALIFIERS}.*", NO_UPS_ENV)

       Execute("cd /ftp/KITS/${UPS_BASE_FLAVOR}/${UPS_PROD_NAME}/${UPS_PROD_VERSION};
          /usr/bin/ln -fs ${UNWIND_PROD_DIR}.* . || true",NO_UPS_ENV)

    action = postdeclare
#
# Make a xxx.ups.tar file
       Execute("test -d \"${UNWIND_UPS_DIR}\" && cd ${UNWIND_UPS_DIR} &&
         tar cf ${UNWIND_PROD_DIR}.ups.tar . || true", NO_UPS_ENV)
end:

group:
    #
    # products installed locally
    options = "local"

common:
    #
    # actual locations of things on local system
    UPS_THIS_DB = "/fnal/ups/db"
    UNWIND_PROD_DIR = "/fnal/ups/${UPS_PROD_NAME}/${UPS_PROD_VERSION}/
      ${UPS_PROD_FLAVOR}${UPS_PROD_QUALIFIERS}"
    UNWIND_UPS_DIR = "${UNWIND_PROD_DIR}/ups"
    UNWIND_TABLE_DIR = "${UPS_THIS_DB}/${UPS_PROD_NAME}"
    UPS_TABLE_FILE = "${UPS_PROD_VERSION}.table"
    #
    # declared values of things
    UPS_PROD_DIR = "${UNWIND_PROD_DIR}"
    UPS_UPS_DIR = "ups"
end:
```

Chapter 32: The UPP Subscription File

UPP stands for **UNIX Product Poll**. It is a layer on top of **UPD** that can be used to facilitate the update of products on a local **UPS** node as new versions become available on a product distribution node. **UPP** is configured on the local node by subscription files, which we describe in this chapter. The functions **UPP** can be configured to perform on a local node include:

- notify the client of new and updated products on a specified distribution node
- perform product installations and updates
- install/update product dependencies and resolve chains to maintain integrity of main product
- delete old product versions

32.1 UPP Subscription File Header

The header of the **UPP** subscription file consists of lines of the form:

```
variable = value
```

in which the following variables may be defined:

`file`	Always set this to the value **`upp`**
`mail_address`	The email address where you want command output to be sent
`dist_node`	The node name of the product distribution node to query for new/updated products
`newprod_notify`	Set to **`T`** (True) if you want to be notified of brand new products; otherwise, leave it out or set it to any other value (e.g., **`F`**)
`data_dir`	The full path to the directory where you want **UPP** to maintain bookkeeping files. Each subscription file must have its own `data_dir`. `data_dir` must be writable when called from **cron**.

32.2 Stanzas

After the header, the **UPP** subscription file consists of one or more stanzas, each bracketed by the lines `begin` and `end`. The number of stanzas per file is not limited. A stanza cannot refer to multiple products, however there can be multiple stanzas for the same product (e.g., for treating different instances of the same product differently). Each stanza has three elements:

- identification of a product or particular instances of a product
- identification of the condition(s) for which you want **UPP** to perform the instructions you give it (done via an *action* statement)
- a list of instructions, or functions to perform, for each condition

32.2.1 Product Instance Identification

The following terms can be used for matching a new or updated product instance on the distribution node:

`product`	Product name
`flavor`	Product flavor
`version`	Product version
`qualifiers`	Product qualifiers
`prod_dir`	Product root directory
`chain`	Product chain

Within a stanza, *all* instances that match a given set of values will be operated on (in contrast to the standard **UPS** and **UPD** matching algorithms; see Chapter 26: *Product Instance Matching in UPS/UPD Commands*). You must at least specify the product name (the product name alone matches all instances), all further specification is optional and used to restrict the set of instances matched. Typically, only product and sometimes flavor are specified.

32.2.2 Conditions and Instructions

After identifying a product instance(s) within the stanza, you need to tell **UPP** what condition(s) to look for regarding the product, and what to do when the conditions are met. One or more `action = <value>` lines can be included to set conditions, each followed by a list of functions to perform.

Actions

In a subscription file, the **`action`** keyword can take the following values (indicating the condition):

`newversion`	A new version of the product is installed on the distribution node.
`<chain>`	The product is chained to *chain*, where *chain* can be current, test, or any other predefined or user-defined chain (see section 1.3.5 *Chains*). E.g., **`action = current`**

List of Functions

The functions that can be used under an `action = <value>` line currently take no arguments. All of the behavior is assumed to be defined by the local **UPD** configuration (described in Chapter 31: *The UPD Configuration File*) when **UPP** is invoked.

`notify`	Place a notice of the new product instance in the mailed output.
`install`	Install the subscribed product via **`upd install`**.
`delete`	Delete existing instance via **`ups undeclare -Y`**.
`reget`	Short for: delete, then reinstall
`update`	Update via **`upd update table_file:ups_dir`**.
`resolve`	Run any **`ups declare`** commands as necessary to make chains match so that parent product and dependencies run properly together.

32.3 Examples

32.3.1 Sample UPP Subscription File

```
FILE = upp
MAIL_ADDRESS = somebody@fnal.gov
DIST_NODE = fnkits.fnal.gov
DATA_DIR = /var/adm/upp
NEWPROD_NOTIFY = T
#
# example of watching for new releases of a particular product:
#
begin
    product = xntp
    flavor = SunOS+5

    action = newversion
       notify
end

#
```

```
# example of a product you want installed, but not chained, when it goes current:
#
begin
    product = ximagetools
    flavor = SunOS+5

    action = current
           notify
           install
end

#
# example of tracking kits closely:
# * when a new version comes out we notify
# * when it is declared or modified as test we reget it, assuming the product
#   is allowed to have internal changes while in "test". We "resolve" to have it
#   declared test here.
# * when it is declared current, we install it (which only does something if we
#   don't have it) and update it to catch re-issues of table files,etc.  We
#   "resolve" to have it declared current here.
#
begin

product = exmh
flavor = SunOS+5

    action = newversion
           notify

    action = test
           notify
           reget
           resolve

    action = current
           notify
           install
           update
           resolve
end
```

32.3.2 A Longer Annotated Example

Here is a sample **UPP** subscription file with one stanza. It is more comprehensive than a typical subscription file, illustrating the use of *all* the supported actions and functions. Explanations are provided line by line.

`file = upp`	This identifies the file as a **UPP** subscription file.
`mail_address = joe@fnal.gov`	Send mail notifications to *joe@fnal.gov.*
`dist_node = fnkits.fnal.gov`	Use `fnkits.fnal.gov` (the central Computing Division distribution node where the `KITS` database resides) as the **UPD** product distribution node to contact
`data_dir = /var/adm/upp`	Use `/var/adm/upp` as the **UPP** bookkeeping directory

`newprod_notify = T`	Yes, notify me of new products appearing on the **UPD** server node (i.e., in the `KITS` database).
`begin`	Begin a stanza.
`product = exmh`	Subscribe to **exmh**. In other words, perform the following actions on it and on its dependencies (the **exmh** flavors and versions remain unspecified in this example, therefore all instances are matched).
`action = newversion`	Define in the following lines one or more functions to perform when a brand new version of **exmh** appears in `KITS`.
`notify`	Send a notification message to *joe@fnal.gov*
`reget`	Remove (via **`ups undeclare -Y`**) and then reinstall (via **`upd install`**) the appropriate instance on the local node, and the necessary dependencies.
`resolve`	**`upd install`** has determined which `ups declare` commands need to be run so that all the chains match up properly for the dependencies to work; run these commands.
`action = current`	Define in the following lines one or more functions to perform when a version of **exmh** is chained to current in `KITS`.
`notify`	Send a notification message to *joe@fnal.gov*
`install`	Install the current instance in `KITS` (and its dependencies as necessary) on the local node
`resolve`	**`upd install`** has determined which `ups declare` commands need to be run so that all the chains match up properly for the dependencies to work; run these commands.
`action = deprecated`	Define in the following lines one or more functions to perform when a version of **exmh** gets deprecated (i.e., chained to a user-defined chain of "deprecated") in `KITS`. This is included to illustrate the use of user-defined chains.
`notify`	Send a notification message to *joe@fnal.gov*
`delete`	Delete the instance on the local node via **`ups undeclare -Y`**.
`end`	End stanza. (Additional stanzas may be included in the same file; use `begin` and `end` to bracket each one.)

Glossary

This glossary defines terminology as it is used in the context of **UPS** and **UPD** v4.

action

Also called a **UPS** action. Actions are used in table files to group together functions that **UPS** must perform when a particular command is issued. An action consists of an ACTION=VALUE keyword (e.g., ACTION=SETUP) plus any functions listed underneath it.

active product instance

The product instance that is currently setup. The *active instance* may be different than the *current instance*.

archive UPS database

A **UPS** database on a product distribution node in which the **UPS** product instances are stored in archive format (e.g., tar, gzip), available for downloading to a user node. Also called a *distribution database*.

bootstrap

(In this manual, we discuss bootstrapping the **CoreFUE** product, which includes **UPS**, **UPD** and **perl**.) Install **UPS**/**UPD** on a machine on which no prior versions of these products are installed.

build

The process by which a distributable instance of a software product is constructed. The build procedure results in a unique combination of product name, version, flavor, and qualifiers. The actual process varies by product and by developer. It can simply consist of a set of copy commands, or be as sophisticated as generation of executables from a master source library of the software.

chain

A chain is a **UPS** database entry (in a chain file) that points to a declared product instance, tagging the product instance according to its release status (e.g., current, test). Chains allow users to specify the version of a product according to its status, rather than by its version number. The defined chain names are: *current*, *test*, *development*, *new*, and *old*. Their corresponding options (or flags) used in commands are: **`-c`**, **`-t`**, **`-d`**, **`-n`**, **`-o`**. The **`-g <chainName>`** option allows definition of an arbitrary chain name.

Chains are set by the **`ups declare`** command; hence the term *declare a product instance as current*.

chain file

Chain files reside in the product-specific directory under the **UPS** database directory, and maintain the chain information. Chain files are named according to the chain name, and end with `.chain`, e.g., `current.chain`. A chain file's contents is simply the list of the product instances (specified via sets of keyword/value pairs) that have been declared with that chain.

cluster

For the purposes of this document, a cluster is set of CPU nodes which share one or more **UPS** databases and product areas. Generally the nodes of a cluster also share (at least) login areas.

configure a product instance

For any product instance that requires configuration, an ACTION=CONFIGURE line is provided in its table file, with functions listed beneath it. In **UPS** *configuring a product instance* means executing these functions by issuing the **`ups configure`** command with appropriate options. This happens by default when a product is declared, otherwise it can be run manually. The functions perform all the configuration needed for the product to run, minus that which requires input from the installer (see *tailor a product instance* and *INSTALL_NOTE* for that portion).

coreFUE

A bundle of **UPS**, **UPD** and **perl**, the core pieces of the Fermi UNIX Environment.

current instance (of a product)

A product instance that is declared as current in the database (i.e., to which the chain "current" points). The current instance of a product is the default for **UPS** and **UPD** commands when no version or chain is specified. For a given product, there may be one current instance each for several flavor/qualifier pairs.

daemon process

A background process that is configured to start up automatically on a system at boot time and to stop at shutdown.

database

See *UPS database*.

database configuration file

The **UPS** database configuration file contains system-specific information that customizes the **UPS** installation on a node or cluster. If it exists, it must reside under the database directory in the file `/path/to/ups_database/.upsfiles/dbconfig`.

declare a product instance to UPS

The **`ups declare`** command makes a product instance known to the **UPS** database and accessible by **UPS**. Declaration does not by itself make the product instance usable since any product requirements (and often other conditions) must also be satisfied, but declaring the product instance is a prerequisite for use (unless you're using **UPS** products without a database).

declare a product instance current

Declaring a product instance as "current" essentially tags it as the default instance (when its flavor/qualifiers are matched). The declaration creates a current chain file or chain file entry that points to the version file for the instance. Product instances can also be declared as test, development, new or old, or as a user-defined chain for easy access.

declared product instance

An instance of a product which has been declared to a **UPS** database.

default function

The functions (as listed in section 34.3 *Function Descriptions*) that a **UPS** command completes (in addition to its internal processes) if no corresponding ACTION=COMMAND keyword line is found in the matched table file, or if the function **`doDefaults([<ACTION>])`** is listed under the corresponding ACTION=COMMAND keyword line. Only the commands **`setup`** and **`unsetup`** actually have default functions.

dependencies

Additional products that must be installed, declared, and setup to ensure the successful operation of a given product or to enable special features within it. When a product instance is setup, its dependencies also get setup by default.

distribution database

A **UPS** database in which **UPS** product instances are available for distribution to user nodes. A distribution database may be in archive or live format. The default distribution database at Fermilab is `KITS` which is maintained by the Computing Division on the node *fnkits.fnal.gov.*

distribution node

This term is used in **UPD** to refer to the node on which **UPS** products are stored and available for distribution to user nodes. A distribution node contains a distribution **UPS** database (can be live or archive) and a distribution products area, and runs **UPS**, **UPD**, a Web server and an **FTP** server (preferably **WU-FTP**). It is sometimes called a *server node*.

It is possible to maintain a distribution database on one machine running **UPS** and **UPD** and a Web server, and maintain the corresponding distribution products area(s) on a different one running an **FTP** server, if the machines share a file system.

end user

Anyone who uses **UPS** products, but does not install, update, maintain, or develop them.

FermiTools

FermiTools are Fermilab-developed software packages that are believed to have general value to other application domains, and thus have been made publicly available in a special subdirectory of `KITS` via anonymous **FTP** and **www**. They do not require **UPS**. Installation and use instructions come with each product.

Fermi UNIX Environment (FUE)

FUE started as a project for providing a cross-department, cross-division structure for the proposal, discussion, design and implementation of all things that affect the user when operating in a UNIX environment at Fermilab. Currently it consists of scripts and programs that form a uniform UNIX environment, standards documents, and the **UPS** suite of tools (see `http://www.fnal.gov/cd/FUE/`).

flavor

To indicate the operating system (OS) dependency of a product instance, we use the term *flavor.* This extra term allows us to differentiate by operating system, and optionally OS version, while maintaining the same product name and version number for separate instances. Some products do not require customizing for the different operating systems (typically those without compiled code), but most do and therefore come in several flavors.

flavor table

A list of a machine's flavor including every level of specificity that you could use to find or declare a product instance. For example, on a SunOS+5.6 machine, the complete flavor table reads:

```
SunOS+5.6
SunOS+5
SunOS
NULL
ANY
```

FTP server node

As regards **UPD**, this node contains **UPS** product instances (and files associated with them) that may be downloaded to a user node, and it runs an **FTP** server. Usually it is the same node as the Web server node, and called simply the *server node* or the *distribution node*.

FUE

See *Fermi UNIX Environment*.

fullFUE

A bundle of **coreFUE** plus the pieces which are strongly recommended for on-site systems: **systools**, **shells** and **futil**.

function

A **UPS**-defined entity used in table files that executes an operation within an action. The supported functions are listed in section 34.3 *Function Descriptions*. One or more functions always follow an ACTION=VALUE keyword line.

A function is specified in a shell-independent manner, but contains enough information to allow it to be transformed into a **sh** or **csh** family command (e.g., **`sourceRequired()`**, or **`execute()`**), or to be interpreted directly by **UPS** (e.g., **`writeCompileScript()`**).

install a product instance

Copy a product instance to a local system from another location (usually from a distribution node) and perform the necessary steps to make it work.

INSTALL_NOTE

A file that describes procedures that the installer must perform manually to complete the installation of a product. This file is provided by the product developer as needed.

instance

See *product instance*.

internal processes (or internals)

The set of processes that a **UPS** command completes, regardless of the contents of the product instance's table file. The internal processes are driven by the command line parameters and options, and relevant environment variables.

keyword

Keywords are used in the **UPS** database files. They are essentially parameters to which values must be assigned. The supported set of keywords listed in section 27.4 *List of Supported Keywords* collectively contains the information **UPS** requires for managing a **UPS** installation and all its **UPS** products. Some of the keywords can be used in all the **UPS** product management file types, others are restricted to certain file types.

keyword value

The value assigned to a keyword in one of the **UPS** database files.

KITS

The name of the **UPS** product distribution database on the central product distribution node at Fermilab, *fnkits.fnal.gov*. The location of the `KITS` database is `/ftp/upsdb`. **UPS** products are stored in the corresponding product area, `/ftp/products` (symlinked to `/ftp/KITS`), as tar files, generally. **UPD** commands access the `KITS` database and products area by default.

live UPS database

A **UPS** database in which the **UPS** product instances are unwound, i.e., not stored in archived format (e.g., tar, gzip).

local UPS database

A live **UPS** database on a local node. For user nodes, a database in which **UPS** product instances are declared and available to be accessed and used.

local user node

See *user node*.

make

The UNIX **make** utility is a tool for organizing and facilitating the update of executables or other files which are built from one or more constituent files. See *UNIX at Fermilab* or a standard UNIX reference text for more information.

Makefile

First, see *make* above. A Makefile is a blueprint that you design and that **make** uses to create or update one or more target files (usually executables) based on the most recent modify dates of the constituent files. See *UNIX at Fermilab* or a standard UNIX reference text for more information.

operating system (OS)

A control program for a computer that allocates computer resources, schedules tasks and provides the user with a way to access the resources. See document *DR0010* in the Computing Division Web pages for the latest information on supported UNIX operating systems at Fermilab.

operating system version (OS version)

Like other software, an operating system gets fixed and enhanced periodically, and is released by the vendor with a new version number (e.g., IRIX 5.1, IRIX 5.2). Sometimes **UPS** products must be changed to continue to work properly under a new operating system version.

operating system type (OS type)

The name of the basic operating system, without release number, as returned by the command `ups flavor -2` (for example IRIX or SunOS).

overlay

An overlaid product gets distributed and maintained in the product root directory of its main product. The set of products overlaid on a main product is collectively referred to as *the overlay.*

parent product

A dependency's *parent product* is that for which it is a dependency. A product may have multiple parent products.

platform

Platform technically refers to the machine type (hardware) of a computer system. However, since until quite recently in the UNIX world there has been a near-perfect correspondence between hardware platform and OS type (e.g., Digital Alphastations run OSF1), sometimes platform is used loosely to refer to the OS type. This correspondence is changing as Linux can be run on PC, Digital, Sun and IBM hardware.

process an action

UPS converts the shell-independent functions listed underneath an ACTION keyword line in a table file into code appropriate to the shell, and writes the output to a temporary file. This is call processing an action.

product

See *UPS product*

product developer

A person who develops and maintains software products, and makes them available for distribution by installing and declaring them to the `KITS` or other distribution database. Sometimes called a product maintainer.

product installer

A person who downloads **UPS** products from a distribution node (through **UPD**, **UPP** or **FTP**), installs them on a local system, and declares them to a local **UPS** database (often the local system administrator acts as the product installer).

product instance

The term *product instance*, or just *instance*, is used to represent a copy of a product, namely a unique combination of product name, version, flavor and qualifiers within a **UPS** database. For a given product, multiple instances may exist in the database to allow users a choice of version and/or flavor/qualifier pair. A product instance may be chained; hence the term "the current instance of a product".

product name

The name of a **UPS** product as it appears in its **UPS** database files.

product root directory

The directory in which a product instance (i.e. its executables) and (optionally) its associated files reside. The product instance generally has a directory structure of its own, starting at this root directory. Each instance of a product has a separate product root directory.

product user

See *end user.*

product version

The net result of any change to an existing product is that a new *version* of the product is created; it is still the same product, but it will usually run a little differently. The versions of a product are tracked by version numbers, e.g., v1_0, v1_1, etc. **UPS** allows for multiple versions of a given product to be accessible concurrently to end users.

PRODUCTS (or $PRODUCTS)

The environment variable that points to the **UPS** database(s) on your system. If multiple **UPS** databases exist, $PRODUCTS can be reset in your login files to a colon-separated list of databases.

<PRODUCT>_DIR (or $<PRODUCT>_DIR)

PRODUCT here is the name of a product in upper case (e.g., EMACS_DIR). This is the environment variable that points to the product root directory of the active instance of a particular product; it gets set when the **`setup`** command is run.

qualifier

The product developer may include information about options used at compilation time (e.g., `debug` or `optimized`) or other qualifying information for easy identification of special compilations. This information is declared in the form of *qualifiers*. Qualifiers, when present, are part of the unique instance identification along with product name, version and flavor.

read-only variable

UPS sets several read-only variables that can be used in functions in table files. Many of them correspond to keywords set in the **UPS** configuration file. There is another set of read-only variables available for use in setting location definitions in the **UPD** configuration file.

root directory for product

See *product root directory.*

setup

Each installed, declared **UPS** product instance requires that the **`setup`** command be issued prior to use (unless it is a dependency of one that is already setup). **`setup`** performs the necessary operations in your login environment to make an installed, declared product accessible to you. Typically, the operations include modifying environment variables or adding to your $PATH. Any dependencies defined for the product get setup by default at the same time.

table file

Table files contain non-system-specific and non-shell-specific information that **UPS** uses for installing, initializing, and otherwise operating on product instances. That is, information pertinent to one or more product instances, independent of the installation machine. Table files are provided by the product developer as needed.

tailor a product instance

Tailoring is the aspect of the product implementation that requires input from the product installer (e.g., specifying the location of hardware devices for a software driver package). If the product requires tailoring, a file is usually supplied in the format of an interactive executable (script or compiled binary), and it is run by issuing the **`ups tailor`** command with appropriate options. To *tailor a product instance* means to run this action, and hence, run the file.

tar

The **tar** (tape archive) utility can create, add to, list, and retrieve files from an archive file.

tar file

A tar file is in archived format, and must be unwound for use. **UPS** products are generally stored in `KITS` as tar files.

unknown command handler

A **UPS** feature that allows user-defined actions (e.g., ACTION=XYZ followed by **UPS**-supported functions) in table files that can be run via a corresponding **UPS**-style command (e.g., **`ups xyz [<options>] <product> [<version>]`**)

unsetup

`unsetup` generally undoes the changes to the user's software environment made by **`setup`** in order to make the product no longer available for use. Any dependencies get deactivated automatically at the same time by default.

UPD - Unix Product Distribution

A companion product to **UPS** which provides the functionality for uploading/downloading products between local systems and product distribution servers.

UPD commands

Any of the commands supported by **UPD**. They are listed and described in Chapter 23: *UPD/UPP Command Reference*. These include commands to retrieve **UPS** products or certain individual files or directories from a distribution database, and commands to manage products within a distribution database.

UPP - Unix Product Poll

A layer on top of **UPD** that allows a client to request notification of changes in a distribution node database and to download pre-specified products. **UPP** can be automated. This is a useful tool for keeping abreast of changes/enhancements to your favorite products.

UPS - Unix Product Support

UNIX Product Support (**UPS**) is a software support toolkit which provides a methodology for creating/managing all the UNIX products provided and/or supported by the Computing Division, and a uniform interface for accessing these products. **UPS** is itself a product that must be installed on any machine that will be used to run other **UPS** products.

UPS has two parts: one or more databases which function as a central repository of information about the products, and a set of procedures/programs to manipulate the database(s).

UPS action

See action.

UPS commands

Any of the commands supported by **UPS** to manage products in a **UPS** environment. They are listed and described in section Chapter 22: *UPS Command Reference*.

UPS database

A directory that functions as a repository of information about all the installed, accessible **UPS** product instances on a system. **UPS** allows multiple installed and declared instances of each product. The database contains files for each product which store pointers to and information about the declared instances of the product.

ups directory (or ups subdirectory)

A directory that may contain miscellaneous important files for a product instance; e.g., its table file, scripts that the table file needs to execute, and so on. This directory may reside anywhere; it often resides directly under the product instance's root directory. Not all products have `ups` directories.

UPS product

Software products distributed and managed by the **UPS** system are called **UPS** products. **UPS** products include Fermilab-written programs, a wide range of public domain software, and a host of third party licensed (proprietary) products. **UPS** products are available for distribution in the `KITS` database on *fnkits.fnal.gov.*

user node

A node from which users can run **UPS** products; usually contains a live local **UPS** database and locally-installed products.

version

For a product see *product version*; for an operating system see *operating system version*.

version file

A version file contains system-specific information for each instance of a **UPS** product. One *version file* must exist in the product-specific directory under the **UPS** database directory for each version of a product that is declared to the **UPS** database. The name of the version file is the version number followed by `.version`, e.g., `v2_2.version`.

Web server node

As regards **UPD**, this node contains one or more distribution databases and runs a Web server, and **coreFUE**. Usually it is the same node as the **FTP** server node, and called simply the *server node* or the *distribution node*.

Index

Symbols

"-?" option 2-1, 10-1
+ argument for -K option 2-3
.updfiles directory 1-6
.upsfiles directory 1-6
/etc/init.d directory 14-5
/etc/rc*.d directories 14-5
/usr/local/ area
    Fermilab policy regarding use of 15-2, 15-3
@ symbol 27-8
    use with keywords 22-46
_UPD_OVERLAY keyword 16-7, 27-11
    description 27-8

Variables

$<PRODUCT>_DIR variable 34-18
    as set during setup 22-5
    description 22-5
${DASH_PROD_FLAVOR} read-only variable 31-4
${DASH_PROD_QUALIFIERS} read-only variable 31-5
${PROD_DIR_PREFIX} read-only variable 31-5
${PRODUCTS} read-only variable
    comparison to PRODUCTS env variable 34-19
    description 34-19
${SUFFIX} read-only variable 31-5
${UPD_USERCODE_DB} read-only variable 3-4
${UPD_USERCODE_DIR} read-only variable 3-4
${UPS_BASE_FLAVOR} read-only variable 31-4
${UPS_COMPILE} read-only variable
    description 34-19
${UPS_EXTENDED} read-only variable
    description 34-19
${UPS_OPTIONS} read-only variable
    description 34-19
${UPS_ORIGIN} read-only variable
    description 34-19
${UPS_OS_FLAVOR} read-only variable
    description 34-19
${UPS_PROD_DIR} read-only variable
    description 34-19
${UPS_PROD_FLAVOR} read-only variable 31-4
    description 34-19
${UPS_PROD_NAME} read-only variable 31-4
    description 34-19
${UPS_PROD_QUALIFIERS} read-only variable 31-4
    description 34-19
${UPS_PROD_VERSION} read-only variable
    description 34-19
${UPS_THIS_DB} read-only variable
    description 34-19
${UPS_UPS_DIR} read-only variable
    description 34-20
${UPS_USERCODE_DB} read-only variable 31-4
${UPS_USERCODE_DIR} read-only variable 31-4
${UPS_VERBOSE} read-only variable
    description 34-20
$PATH variable 1-10, 2-10, 22-11
$PRODUCTS variable 1-6, 1-10, 25-4
    as used in UPD commands 26-1
    as used in upd install 5-2
    comparison to read-only ${PRODUCTS} 34-19
    for multiple databases 25-2
    use in database selection 26-1
    use with private database 11-9
    with AFS database 12-4
$SETUP_<DIR> variable 34-18
    as set during setup 22-5
$SETUP_<PRODUCT> variable
    description 22-5
    use with unsetup command 22-6, 22-11
$SETUP_UPS variable 1-10
$TEMPDIR variable
    use with upd addproduct 17-1, 23-7
$UPS_DIR variable 1-10
$UPS_EXTENDED variable
    as set by -e option 24-2
$UPS_EXTRA_DIR variable 12-5
$UPS_OPTIONS variable
    as set by -O option 24-4
$UPS_SHELL variable 1-10

"@" Keywords

@COMPILE_FILE keyword 22-47, 27-9
@PROD_DIR keyword 22-48, 27-9
@TABLE_FILE keyword 22-48, 27-10
@UPS_DIR keyword 22-48, 27-11

A

access.conf file 20-11
accessing a UPS product 2-8, 22-5

accounts
for managing distrib node 20-3
for product installation 11-1, 11-2
ftp 20-3, 20-8
separate by product category 11-2
the products account 11-1
updadmin 20-3, 20-5, 20-12, 21-6
wwwadm 20-3, 20-4, 20-7, 20-8
ACTION keyword
"unchain" names as values 33-3
chain names as values 33-3
description 27-4
detailed 31-5, 33-1
UPS command as keyword value 33-1
use in table files 34-1
user-defined values 33-3
actions
"unchain" name as keyword value 33-3
and "unactions" 33-2
called by other actions 33-4
chain name as keyword value 33-3
examples 34-18
functions used in 34-1
overview 31-5, 33-1
processing of 24-9
reference 33-1
undoing chains in table files 33-3
undoing reversible functions 33-2
UPS commands used as 33-1
use in table files 33-1
use in updconfig 31-5
use with "unknown" commands 33-3
add chain to product on distrib node 17-7, 23-33
add product to distrib node 17-3, 23-3, 23-8
using template_product 18-6
add product to KITS 17-3, 23-3, 23-8
special product registration 17-3
add table file to distrib node 17-5
update for existing product 17-6
add ups directory to distrib node
update to existing product 17-6
addAlias function
description 34-2
AFS
$PRODUCTS variable 12-4
$UPS_EXTRA_DIR variable 12-5
configuring local database 12-2
installing into local database 12-5
installing into local products area 12-4
installing product into AFS product area 8-3
local configuration options 12-1
local FUE initialization files 12-3
products requiring special privileges 12-6
providing access to AFS products 12-1
updating /usr/local/bin 12-6
upsdb_list file 12-2
using AFS UPD and installing locally 8-2
using local database with 12-1, 12-2
AFS database
use with local database 5-3
aliases defined by UPS 1-10
announcement of new/updated product 17-10
anonymous FTP 7-5
download files from fnkits 7-2
apache product
for distrib node web server 20-5, 20-10
apropos command 38-3
ARCHIVE_FILE keyword 22-47
as set by -T option 24-4
description 27-4, 28-2
AUTHORIZED_NODES keyword 22-47, 30-1
as set by -A option 24-1
description 27-4, 28-2
autostart
configuring UPS to allow 14-1
control files 14-3
permissions 14-4
disabling 14-5
installing product for 14-2
START action 14-3
start script example 36-4
STOP action 14-3
stop script example 36-5
TAILOR action 14-3
ups script 14-1
ups_shutdown script 14-1, 14-2
ups_startup script 14-1, 14-2

B

bin directory of product 16-1, 16-3, 16-5, 18-4, 19-1
description 15-6
bootstrapping CoreFUE
bootstrap script 13-1, 13-5
config.custom file 13-2
configurator script 13-2
customizing configuration 13-3
log file 13-5
predefined configurations
for NT 13-2
for UNIX 13-1
running the procedure 13-5
sample customization 13-4
space requirements 13-1
stage1.sh file 13-1, 13-5
stage2.sh file 13-5
user defined configurations 13-2
user-customized configuration 13-2

C

catman directory 15-7
CATMAN_SOURCE_DIR keyword 22-47
description 27-4
CATMAN_TARGET_DIR keyword 22-47, 30-1
description 27-4
CD-ROM
product distribution 20-14
setup product directly from 22-7
chain
adding product to distrib node 17-3, 23-7
as action in table files 33-3
change (on declared instance) 10-7
current 1-4
declare at product declaration 3-6, 10-2
declare to installed instance 10-4

definition 1-4
development 1-4
new 1-4
old 1-4
remove and add new 10-7
remove from instance 10-6
specification in command 25-1
test 1-4
usage 1-5
use in instance matching 26-3
user-defined 1-4
chain files 1-6, 22-79, 29-1
and product removal 10-7
creating 29-1
description 29-1
examples 29-3
information storage format 29-1
instance matching within 26-3
keywords 29-1
overview 27-1
CHAIN keyword 22-47
description 27-4, 29-2
chain names 1-5
chain options 1-5
change a chain 10-7
change product chain on distrib node 17-7
command defaults 1-8
command output formats for ups list 24-7
command syntax 1-8
description 25-1
comment solicitation INT-5
COMMON: keyword 35-3
description 27-4
use in table files 35-3
use in updconfig file 31-2
COMPILE action 37-1
compile script 37-1
COMPILE_DIR keyword 22-47, 27-9
description 27-4, 28-2
COMPILE_FILE keyword 22-47, 27-9
as set by -b option 24-1
description 27-4, 28-2
config.custom file 13-2
configurator script 13-2
configure a product instance 3-9, 22-13
in AFS space 8-5
CONFIGURE action 10-8, 22-80, 36-1
configure script 36-1
for prebuilt binaries 16-5
configuring distribution node 20-1
conventions, notational INT-3
copy a product declaration 22-19
CoreFUE
and AFS 12-1
bootstrapping 13-1
components 12-4, 12-5, 13-1
customizing configuration 13-3
local installation on AFS machine 12-4
predefined configurations
for NT 13-2
for UNIX 13-1
running the bootstrap procedure 13-5
sample bootstrap customization 13-4
space requirements 13-1
user defined configurations 13-2
courtesy links to initialization files 1-9
create a database
checklist for preparation 11-9
on machine running AFS 12-2
cron
use to automate UPP 4-3, 6-4
CURRENT action 36-3
current chain 1-4
as default 1-8
current script 36-3
CVS 17-9
use with template_product 18-8
CYGWIN
bin directory 11-8
perl version 11-7
UPS/UPD installation issues 11-7

D

database (See UPS database)
database configuration file (See UPS configuration file)
database files
chain files 29-1
included comments 27-3
keywords 27-1
location 11-6
ownership 11-3
permisisons 11-3
pointers to directories 11-6
syntax 27-3
UPD configuration file 31-1
UPP subscription file 32-1
UPS configuration file 30-1
version files 28-1
database on distrib node
file permissions 20-7
host-based access restriction 20-6
user-based access restriction 20-6
database selection algorithm 5-2, 26-1
database specification in commands 25-4
dbconfig file (See UPS configuration file)
dbconfig.template file 30-1
listing 30-2
declare a chain to an instance 3-6, 10-2, 22-21
declare a product 3-5, 10-1, 22-21
after download via FTP 3-5, 10-1
as part of installation 5-1
declare chain at same time 3-6, 10-2
node/flavor-specific functions present 10-4
specifying ups dir and table dir 3-5, 10-2
to local database 7-4
DECLARED keyword 10-6, 22-47
description 27-4, 28-2, 29-2
DECLARER keyword 10-6, 22-47
description 27-4, 28-2, 29-2
defaults for UPS/UPD commands 1-8
Also see command reference chapters
delete product component from distrib node 17-8
delete product from distrib node 17-8
using template_product 18-8
dependencies
and unsetup command 22-11
conflict resolution 35-4

cross-database support for 1-5
database selection for install 5-3
definition 1-5
finding them for a product 2-7
list using ups depend 2-7
multiple levels of 1-5
non-UPS products 35-4
on distribution node, list using upd depend 4-5
order of product setups 35-5
setupOptional function in table file 35-4
setupRequired function in table file 35-4
dependency matching 26-2
DESCRIPTION keyword 22-47
description 27-4, 28-2, 29-2
determine if product update needed
using upd install -s 10-13
using upd update -s 10-13
using upp 10-13
development chain 1-4
use during product development 16-2
distributing UPS products
announcement policies for new products 17-10
overview 17-1
to KITS (checklist) 19-3
to KITS (using template_product) 19-3
distribution node
~ftp area 20-4
access restrictions on database
host-based 20-6
user-based 20-6
configuration and management 20-1
configure and manage 20-1
fnkits.fnal.gov 3-2, 7-2
FTP server 20-1
configuration 20-7
KITS database (on fnkits.fnal.gov) 3-2
limiting product distribution 20-11
nodes other than fnkits 7-4
option_list product description 20-12
reporting on FTP and Web accesses 20-10
response to upd addproduct command 20-2
response to UPD commands 20-1
response to upd install command 20-2
response to upd modproduct command 20-2
restrict downloads from database 20-11
restrict uploads to database 20-11
updconfig pre and postdeclare actions 20-10
user accounts 20-3
web server 20-1
configuration 20-5
doc directory 15-7
documentation for products 15-7
doDefaults function
description 34-3

E

editing database files 10-11
END: keyword 35-3
description 27-4
use in table files 35-3
use in updconfig file 31-2
envAppend function
description 34-3
environment
and usage of command options 25-4
changes made by UPS 1-10
initializing for UPS 1-9
envPrepend function
description 34-4
envRemove function
description 34-4
envSet function
description 34-5
envSetIfNotSet function
description 34-5
envUnset function
description 34-5
examples directory 15-7
exeAccess function
description 34-6
exeActionOptional function
description 34-6
use to call another action 33-4
exeActionRequired function
description 34-6
use to call another action 33-4
execute function
description 31-6, 34-7
use in dbconfig 31-6

F

Fermi UNIX Environment
initializing 1-9
FermiTools INT-2, 4-6, 7-1, 7-2, 21-3
FILE keyword 30-1
description 27-5, 28-2, 29-2
file ownership
considerations 11-3
database files 11-3
product files 11-3
file permissions
configuring UPD to set (product files) 11-2
database files 11-3
extra security 11-3
unwound tar files 11-2
file system semantics
and group ids 11-2
Berkeley 11-2
setting 11-2
System V 11-2
fileTest function
description 34-7
flavor
ANY, as used in flavor matching 26-4
definition 1-3
NULL 1-3
specification in KITS 1-3
FLAVOR keyword 22-47
description 27-5, 28-2, 29-2
value ANY 35-3
flavor levels 2-2, 24-7
flavor of machine, determining 2-1, 22-35

flavor specification
(-f, -H and number options) 1-3
use in instance matching 26-3
flavor table 24-7
definition 2-2, 22-36
flavor.products file 14-3, 14-5
permissions 14-4
fnalonly products 21-3
fnkits.fnal.gov distribution node 4-1
adding products to 23-8
anonymous FTP for downloading products 7-1, 7-2
config file locations 21-6
database location 21-6
directory hierarchy 4-6
FermiTools 4-6, 7-1, 7-2
FTP server log file 21-7
ftpgroups file 21-6
KITS database 3-2
KITS product categories 21-3
product pathnames for FTP access 4-7, 4-8
product permissions 4-6
proprietary products 4-8
registration for downloading products 3-2, 7-2
server maintenance 21-6
using FTP to download products 7-1
web server log file 21-7
formatted ups list output 22-45
FTP
declare product after download 3-5, 10-1
downloading product components 7-1
product installation 7-1, 7-2, 7-5
FTP server
access file 20-11
log file on fnkits 21-7
log searcing 20-13
on distrib node 20-1
ftpaccess file 20-7, 20-11
ftpgroups file 21-6
ftpweblog product 20-10
FUE initialization files
courtesy links to 12-3, 12-5, 12-6
for use with AFS 12-3
functions
addAlias 34-2
case (in)sensitivity of 34-1
doDefaults 34-3
envAppend 34-3
envPrepend 34-4
envRemove 34-4
envSet 34-5
envSetIfNotSet 34-5
envUnset 34-5
examples 34-18
exeAccess 34-6
exeActionOptional 34-6
exeActionRequired 34-6
execute 31-6, 34-7
fileTest 34-7
overview 34-1
pathAppend 34-8
pathPrepend 34-8
pathRemove 34-9
pathSet 34-9
preprocessing via compile script 37-1
prodDir 34-9
reference 34-1
reversible 33-2, 34-1
setupEnv 34-10
setupOptional 34-10
setupRequired 34-10
sourceCompileOpt 34-11
sourceCompileReq 34-11
sourceOptCheck 34-12
sourceOptional 34-13
sourceReqCheck 34-13
sourceRequired 34-14
to be added in future 34-17
translation into shell commands 24-9
unAlias 34-14
unProdDir 34-14
unsetupEnv 34-15
unsetupOptional 34-15
unsetupRequired 34-16
use with ACTION keyword 34-1
writeCompileScript 34-16

G

-g option for user-defined chain 1-5
groff command
ascii output 38-6
-man option 38-1
PostScript output 38-6
GROUP: keyword 35-3
description 27-5
use in table files 35-3
use in updconfig file 31-2

H

hardcoded paths problem 15-4
help on UPS/UPD commands 2-1, 10-1
help online
ups help command 22-41
html directory 15-7
HTML_SOURCE_DIR keyword 22-47
description 27-5
HTML_TARGET_DIR keyword 22-47, 30-2
description 27-5

I

include directory 15-7
independent table file 17-5
Info directory 15-7
INFO_SOURCE_DIR keyword 22-47
description 27-5
INFO_TARGET_DIR keyword 22-47, 30-2
description 27-5
init.d directory
location 14-1
initializing UPS environment 1-9
courtesy links to files 1-9

INSTALL_NOTE file 7-1, 15-6, 19-1
  configuring product 22-15
  mention of node/flavor-specific functions 10-4
  mention of unconfigure actions 10-8
  sample 16-9
installation methods for UPS products, summary 3-1
installer accounts
  choosing 11-1
  file system semantics 11-2
  multiple 11-1, 11-2
  products account 11-1
  separate by product category 11-2
  setting gid 11-1, 11-2
  single 11-1
  UPD configuration issues 11-2
installing a product
  choose whether to declare qualifiers 3-8
  components to download (using FTP) 7-1
  configuring 3-9
  declare manually after FTP download 7-4
  for development/testing 5-3
  interruption during install 3-8
  into AFS space 8-3
  into private database 11-9
  KITS product categories 17-3
  KITS special product registration 17-3
  local install using AFS UPD 8-2
  onto distrib node 17-3
  pass options to local declare 5-2
  procedural checklist when using UPD 5-3
  products requiring special privileges 8-1, 12-6
  root privileges 12-6
  table file product 17-5
  tailoring 3-9, 22-67, 22-69
  troubleshooting 9-1, 10-17
  ups installasroot command 12-6
  using FTP 7-1, 7-2, 7-4
  using UPD 5-1
  using UPP 6-1
  with all dependencies (using UPD) 5-5
  with different name than on server 3-8
  with no dependencies (using UPD) 5-7
  with required dependencies (using UPD) 5-7
instance
  declare a chain for 10-4
  definition 1-4
  determine if update needed 10-13
  determine instance to act upon 26-1
  install and declare 5-1
  specification via chain or version 25-4
  specify multiple ones in command 25-3
  verify integrity of 10-10
instance matching 26-1
  in chain file 26-3
  in table file 26-3
  in updconfig file 31-2
  in version file 26-3
  use of flavor and qualifiers 26-4
instance selection by chain 1-4
instance specification on command line 25-4
internal command processes 24-9

K

-K option
  description for use with ups list 22-46
  keyword arguments 2-3, 22-46
  with upd list 4-2
  with ups depend or upd depend 2-8, 22-30
keywords 27-1, 28-1
  case (in)sensitivity of 27-2
  DECLARED 10-6
  DECLARER 10-6
  definition 27-2
  in ups list output 2-3
  list with descriptions 22-47, 27-3
  list with file types 27-3
  MODIFIED 10-6, 10-13
  MODIFIER 10-6
  overriding values 27-3
  syntax 27-2, 27-8
  use of @ symbol 22-46
  used with -K option in ups list 2-3
  user-defined 27-2
KITS 4-1
  adding products to 23-8
  dbconfig file 21-1
  FermiTools 7-1, 21-3
  fnalonly products 21-3
  product categories 21-3, 23-8
  product registration for special categories 21-3
  proprietary products 21-3
  registration 4-6, 7-2
  updconfig file 21-2
  updconfig pre and postdeclare actions 21-4
  using FTP to download products 7-1
  US-only products 21-3
KITS distribution database 17-3

L

lib directory 15-7
licensed products
  permissions 11-3
link for hard-coded paths 36-2
links to initialization files 1-9
list all current products 22-49
list all fields for a product 2-6, 22-51
list dependencies on distribution node 4-5, 23-15
list product dependencies 2-7, 22-27
list products in database 2-4, 22-49
list products on distribution node 23-31
  use in troubleshooting product installs 9-1
location of database files 11-6
location of product files, considerations 11-4, 11-5

M

man directory 15-7
man -k command 38-3

man page
ascii output 38-6
convert to html 38-6
determine directory for 11-6
file names 38-1
groff 38-1
information categories 38-3
location of files 38-1
nroff 38-1
nroff output file 38-5
nroff source file 16-3, 38-4
PostScript output 38-6
section numbers 38-1
MAN_SOURCE_DIR keyword 22-47
description 27-5
MAN_TARGET_DIR keyword 22-47, 30-2
description 27-5
man2html command 38-6
managing distribution node 20-1
matching product instance
in chain file 26-3
in table file 26-3
in updconfig file 31-2
in version file 26-3
use of flavor and qualifiers 26-4
MODIFIED keyword 10-6, 22-47
description 27-5, 28-2, 29-2
updating 22-71
used to determine if update needed 10-13
MODIFIER keyword 10-6, 22-47
description 27-5, 28-2, 29-2
updating 22-71
multiple databases
adding a private database 11-9
AFS and local 8-2
and your UPD configuration 3-4
configuring UPD for 31-9
database selection algorithm 26-1
default database 1-8
how UPD selects a database 5-2, 26-1
reasons for using 11-6
specifying $PRODUCTS 1-8, 25-2
support for 1-6
-z option for specifying database 24-5

N

new chain 1-4
news directory 15-7
NEWS_SOURCE_DIR keyword 22-48
description 27-6
NEWS_TARGET_DIR keyword 22-48, 30-2
description 27-6
NFS-mounted database
using local database with 12-1
NIS cluster 12-1
node.products file 14-3, 14-5
permissions 14-4
notational conventions INT-3
nroff command 38-4
for man page 16-3
-man option 38-1, 38-5
NULL flavor 1-3
number options (-0 through -3) 2-2, 22-36
usage information 25-4

O

old chain 1-4
online help
ups help command 22-41
option flags
command-specific info in reference chapters
embedded spaces in arguments 25-2
grouping in commands 25-2
invalid arguments 25-3
multiple arguments 25-2
multiple occurrences 25-3
wildcards 25-4
option usage in commands 25-4
option_list product
description 20-12
order of command line elements 25-1
ORIGIN keyword 22-48
description 27-6, 28-2
OS determination using ups flavor 2-1, 22-36
overlaid products 1-6, 16-7, 27-11
overlays 1-6, 16-7, 27-11

P

parent product determination 10-8, 22-79
parse ups list output
in perl 22-52
in sh script 22-53
pathAppend function
description 34-8
pathPrepend function
description 34-8
pathRemove function
description 34-9
pathSet function
description 34-9
perl
parse ups list output in 22-52
version for use with CYGWIN 11-7
permissions
configuring UPD to set for product files 11-2
database files 11-3
extra security 11-3
on downloaded products 3-7
on files created in distrib database 20-7
unwound tar files 11-2
pointers in database files 11-6
pre-built binary products 16-5
inserting into template_product 18-4
pre-build checklist 19-1
PROD_DIR keyword 22-48, 27-9
as set by -r option 24-4
description 27-6, 28-2
PROD_DIR_PREFIX keyword 3-4, 22-48, 27-9, 30-2
description 27-6
prodDir function
description 34-9
product announcement checklist 19-3

product categories in KITS 17-3
default 21-3
FermiTools 21-3
FNAL only 21-3
proprietary 21-3
registration for special categories 17-3
U.S. only 21-3
product dependencies (See dependencies)
product dependency matching 26-2
product development 16-7
announcement policies for new products 17-10
checklist for building product 19-2
checklist for distributing to KITS 19-3
checklist for pre-build 19-1
checklist for product announcements 19-3
checklist for testing 19-2
code management system 16-6
compile script 37-1
configure script 36-1
configure third-party product 16-6
current script 36-3
declaring product during development 16-2
distributing the product 17-1
documentation location 15-7
example procedure for simple product 16-1
man page creation 16-3
overlaid products 16-7
pre-build checklist with template_product 19-1
pre-built binaries 16-5
prep for rebuilding 16-6
read-only variables 34-18
recommendations
fully-specified flavor 15-1
location determination 15-2
nonuse of /usr/local/bin 15-2
nonuse of /usr/local/products 15-3
reproducible build procedure 15-3
self-containment 15-2
shell-independence 15-1
system-independence 15-3
sample directory hierarchy 16-2
selection of build node 16-7
simple build procedure 16-1
start script 36-3
stop script 36-3
table files 35-1
sample 16-2
tailor script 36-3
testing product 16-4, 18-5
third-party products 15-3
uncurrent script 36-3
unflavored scripts 16-4
using template_product 18-1
vendor-supplied products, rebuilding 16-6
product development tools
buildmanager 15-5
CVS 15-5
template_product 15-6
product distribution
announcement policies for new products 17-10
overview 17-1
using template_product 18-1, 18-6
via CD-ROM 20-14
product distribution node (See distribution node)
product documentation 15-7
product files
configure UPD to set location 11-4, 11-5
location 11-4, 11-5
ownership 11-3
permissions 11-3
product flavor 1-3
product installation (See installing a product)
product instance (see instance)
product instance matching (See instance matching)
PRODUCT keyword 22-48
description 27-6, 28-2, 29-2
product registration for KITS 21-3
product removal (See remove a product)
product root directory 15-6
definition 1-3
locate using ups list -K 22-52
simple example of structure 16-2
product use statistics 27-9
product version 1-3
products account 11-1
products area 3-4
adding a new one 11-9
as set in UPD config 3-4
choosing location 11-4
defining during UPS bootstrap 13-2
for development/testing 11-9
for KITS 21-1
PROD_DIR keyword 27-6, 28-2
PROD_DIR_PREFIX keyword 27-6
structure of product root directory 15-6
unwind product tar files into 7-3
products for use only at FNAL 21-3
products for use only in U.S. 21-3
products requiring build 16-6
build script recommendations 15-3
inserting into template_product 18-4
pre-build checklist 19-1
proprietary products 21-3
on fnkits 4-8

Q

qualifiers
choosing whether to declare them 3-8
description 24-8
mixing required and optional 24-9
optional 24-9
overview 1-4
required 24-8
use in instance matching 26-4
QUALIFIERS keyword 22-48
description 27-6, 28-3, 29-2

R

reader comment solicitation INT-5
README file 7-1, 15-6, 19-1
sample 16-8
read-only variables 34-18
PRODUCTS 34-19
to be added in future 34-21
UPS_COMPILE 34-19

UPS_EXTENDED 34-19
UPS_OPTIONS 34-19
UPS_ORIGIN 34-19
UPS_OS_FLAVOR 34-19
UPS_PROD_DIR 34-19
UPS_PROD_FLAVOR 34-19
UPS_PROD_NAME 34-19
UPS_PROD_QUALIFIERS 34-19
UPS_PROD_VERSION 34-19
UPS_THIS_DB 34-19
UPS_UPS_DIR 34-20
UPS_VERBOSE 34-20
rebuilding product 16-7
registering products for KITS 21-3
RELEASE_NOTES file 19-1
sample 16-9
remove a product 10-7, 22-79
unconfiguring 10-9
using UPP 10-8, 10-10
using ups undeclare command 10-8, 22-77
remove a product component
from distrib node 17-8
remove access to product 2-10, 22-11
remove product from distrib node 17-8
using template_product 18-8
retrieve file or dir from distribution node 10-15
retrieve product from distribution node 5-1
reversible functions 33-2
definition 34-1

S

searchlog.cgi script 20-13
selecting database for dependency install using UPD 5-3
selecting database for product install using UPD 5-2
setup command 1-1, 2-8, 22-5
associated environment variables 22-5
for chained instance 2-9
for current instance 2-9
for unchained instance 2-9
reference 22-3
special options 2-9
test if setup would succeed 10-16, 22-33
use in troubleshooting problem installations 9-1, 10-17
-v option for use in troubleshooting 9-1, 10-17
setupEnv function
description 34-10
setupOptional function
description 34-10
use to define dependencies 35-4
setupRequired function
description 34-10
use to define dependencies 35-4
setups.[c]sh files 1-9
courtesy links to 12-3
determine directory for 11-6
pointers to 11-6
SETUPS_DIR keyword 22-48, 30-2
description 27-6
sh
parse ups list output in a scipt 22-53
shell script products
inserting into template_product 18-4
pre-build checklist 19-1
simulate command 9-1, 10-17
source code
revision tracking 17-9
storage in CVS 17-9, 18-8
sourceCompileOpt function
description 34-11
sourceCompileReq function
description 34-11
sourceOptCheck function
description 34-12
sourceOptional function
description 34-13
sourceReqCheck function
description 34-13
sourceRequired function
description 34-14
src directory 15-7
stage1.sh file 13-1, 13-5
stage2.sh file 13-5
stanzas
table file 35-1
UPD config file 31-1
UPP subscription file 6-1, 32-2
START action 36-3
start script 14-3, 36-3
statistics
how to gather 11-10, 27-9
output 27-10
STATISTICS keyword 22-48, 30-2
as set by -L option 24-3
description 27-6, 28-3
detailed description of use 27-9
output from 27-10
STOP action 36-3
stop script 14-3, 36-3
subscription file for UPP
creating 6-1
reference 32-1
sample for product installation 6-3
SUFFIX keyword 20-9, 20-10
syntax of UPS/UPD commands 1-8, 25-1

T

table files 1-6
compile script used with 37-1
detailed description 35-1
examples
action present for some instances only 35-8
execute one action or another 35-8
grouping 35-6
use of FLAVOR=ANY 35-6
with user-defined keywords 35-7
grouping information in 35-3
information storage format 27-2
instance matching within 26-3
keywords 27-2
locate using ups list -K 22-52
location specification 28-5
naming 35-1

ordering elements in 35-3
overwrite 10-14
read-only variables available for use in 34-18
recommendations to developers 35-2
sample for simple product 16-2, 16-4
stanzas 35-1
structure and contents 35-2
test if needs update 10-13
undoing reversible functions 33-2
-V option for debugging 24-9
TABLE_DIR keyword 22-48
description 27-6, 28-3
TABLE_FILE keyword 22-48, 27-10
description 27-6, 28-3
tailor a product instance 3-9, 22-69
TAILOR action 3-9, 22-67, 22-69, 36-3
tailor script 36-3
tar file creation
by upd addproduct 17-1, 23-7
using template_product 18-5
template_product 15-6, 17-2
adding build instructions 18-4
to top-level Makefile 18-4
checklist for building product 19-2
checklist for distributing to KITS 19-3
checklist for pre-build 19-1
cloning 18-2
customizing product tar file 18-5
downloading 18-2
editing top-level Makefile 18-3
inserting pre-built binaries 18-4
inserting product requiring build 18-4
inserting shell scripts 18-4
inserting your product 18-4
Makefile (top-level) 18-3
overview 18-1
removing product from distrib node 18-8
running a build procedure 18-4
temporary script
prevent deletion 24-9
test chain 1-4
test directory 15-7
testing products 18-5
checklist 19-2
third-party products 15-3
toInfo directory 15-6
toman directory 15-6

U

umask 3-7
unAlias function
description 34-14
unchain
as action in table files 33-3
replace chain on distrib node using upd modproduct 17-7
use ups undeclare to remove chain 10-6, 22-77
UNCONFIGURE action 10-9, 22-75, 36-1
unconfigure script 36-1
UNCURRENT action 36-3
uncurrent script 36-3
undeclare a chain 10-6, 22-77
undeclare a product instance 10-7, 22-79
using UPP 10-8
using ups undeclare command 10-8
undoing chains in table files 33-3
unflavored scripts 16-4
UNIX Product Distribution
overview 1-1
UNIX Product Poll 32-1
overview 1-1
UNIX Product Support
overview 1-1
unknown command handler
description 33-3
unProdDir function
description 34-14
unsetup command 2-10, 22-11
$SETUP_UPS variable 1-10
behavior with dependencies 22-11
reference 22-9
use of $SETUP_<PRODUCT> variable 22-6, 22-11
unsetupEnv function
description 34-15
unsetupOptional function
description 34-15
unsetupRequired function
description 34-16
UNWIND_ARCHIVE_FILE keyword 20-9, 20-10
description 27-6
use in updconfig 31-4
UNWIND_PROD_DIR keyword 3-4
description 27-7
use in updconfig 31-3
UNWIND_TABLE_DIR keyword
description 27-7
use in updconfig 31-4
UNWIND_UPS_DIR keyword
description 27-7
use in updconfig 31-3
UPD
command syntax 1-8
configuration file
info for installers 3-3
overriding default 3-4
reference 31-1
overview 1-1
procedural checklist for installation 5-3
upd addproduct command
adding table file product 17-5
adding typical product 17-3
chains 17-3, 23-7
detailed functions 20-2
internal processes 23-8
reference 23-3
response of distrib node 20-2
tar file creation 17-1, 23-7
upd cloneproduct command
reference 23-11
UPD commands
defaults 1-8
dependency matching 26-2
instance matching 26-1
interaction with distrib node 20-1
option flag grouping 25-2
option usage 25-4
order of command line elements 25-1

specifiying version/chain 25-1
specifying multiple products 25-3
UPD configuration file 27-3
AFS issues 8-2
distrib node 20-9
KITS database pre and postdeclare actions 21-4
pre and postdeclare actions 20-10
examples 31-7
AFS 31-10
distrib node config 31-10
distribution from fnkits 31-8
mulitple dbs and distrib nodes 31-9
for KITS database 21-2
info for installers 3-3
organization 31-1
overriding default 3-4, 31-1
overview 27-1
pre and postdeclare actions 31-5
product matching 31-2
reference 31-1
required location definitions 31-3
sample location definitions 31-5
setting file permissions 11-3
stanzas 31-1
upd delproduct command 17-8
reference 23-13
upd depend command 4-5
reference 23-15
upd exist command 10-16
reference 23-17
upd fetch command 10-15
reference 23-19
upd get command
reference 23-23
upd install command 5-1
database selection 5-2
database selection for dependencies 5-3
detailed functions 20-2
-G (pass options to local declare) 5-2, 23-28
internal processes 23-29
procedural checklist for installation 5-3
reference 23-25
response of distrib node 20-2
summary of functions it performs 3-1
syntax and commonly used options 5-1, 23-25
use to determine if product update needed 10-13
upd list command 4-1
reference 23-31
upd modproduct command 17-6, 17-7
reference 23-33
response of distrib node 20-2
upd move_archive_file script 20-2
upd moved_ups_dir script 20-2
_UPD_OVERLAY keyword 16-7, 27-11
description 27-8
upd repproduct command
reference 23-39
upd update command 10-13, 10-14
reference 23-41
upd verify command
reference 23-45
upd.cgi script 20-2, 20-11
access restrictions 20-6
description 20-5
UPD_USERCODE_DB keyword 22-48
description 27-7
UPD_USERCODE_DIR keyword 3-4, 22-48, 30-2
description 27-7
on fnkits 21-2
update product
determine if update needed 10-13
using UPD 10-13
using UPP 10-13
updconfig file (see UPD configuration file)
updconfig.template file 31-1, 31-7
updusr.pm file 31-1
upgrading UPS installation 11-8
UPP
automate upp command via cron 6-4
command syntax 6-4
monitor products on distribution node 4-3
notification of update needed 4-3, 10-13
overview 1-1
remove a product 10-8, 10-10
subscription file
creating 6-1
definition 4-3
sample for product installation 6-3
uses 32-1
upp command 4-3, 6-1
automation via cron 6-4
reference 23-47
syntax 4-4, 6-4
UPP subscription file
adding instructions 32-2
available functions 32-3
creating 6-1
definition 4-3
header description 32-1
instance matching 32-2
reference 32-1
sample 32-3
sample for product installation 6-3
stanza description 32-2
UPS
aliases defined 1-10
benefits of methodology 1-2
chains 1-4
command syntax and defaults 1-8
database 1-1
database directory specification 1-10
motivation for methodology 1-2
multiple database support 1-1
multiple product flavor support 1-3
multiple product version support 1-2, 1-3
overview 1-1
pointer to product root directory ($UPS_DIR) 1-10
product instance 1-4
product version 1-3
products distributed and managed by 1-3
upgrading your UPS installation 11-8
use without a database 1-7, 11-7
UPS commands
"-?" for usage information 2-1
"uncommands" as action keyword values 34-1
as ACTION keyword 33-1
database selection 26-1
defaults 1-8
dependency matching 26-2

instance matching 26-1
keeping statistics on 11-10, 27-9
option flag grouping 25-2
option usage 25-4
order of command line elements 25-1
specifying multiple products 25-3
specifying version/chain 25-1
UPS configuration file 27-3
defining directory locations in 11-6
for KITS database 21-1
for local database on fnkits 21-1
keywords used in 30-1
overview 27-1
reference 30-1
sample 30-2
ups configure command 3-9
reference 22-13
ups copy command 22-19
reference 22-17
UPS database
$PRODUCTS variable 1-10
$UPS_EXTRA_DIR variable for AFS 12-5
.updfiles subdirectory 1-6
.upsfiles subdirectory 1-6
checklist for creating a database 11-9
choosing single or multiple 11-6
configuring local to work with AFS 12-2
create a private database 11-9
create local database to work with AFS 12-2
declare a product instance to 3-5, 10-1
declaring products into local (not AFS) 12-4
definition 1-6
for development/testing 11-9
installing products into local (not AFS) 12-5
list all current products in 2-4
list product information 2-2
listed in upsdb_list file 12-2
multiple (See multiple databases)
NFS mounted 12-1
permissions for files (distrib node) 20-7
providing access to multiple databases 12-2
setting up your own 5-3
with AFS 12-2
standard naming conventions for use with AFS 12-2
structure and contents 1-6
using AFS and local 5-3
using UPS without a database 1-7, 11-7
UPS database files 1-6
chain files 1-6, 29-1
check for inconsistencies 10-10
editing 10-11
keywords 27-1
overview 27-1
UPD configuration file 31-1
UPS configuration file 30-1
version files 1-6, 28-1
ups declare command 3-6, 10-3
as used internally by upd install 5-2
reference 22-21
specifying database 3-5, 10-2
specifying table file path 3-5, 10-2
specifying ups directory 3-5, 10-2
syntax and common options
for declaring chain 10-4
for declaring instance 3-5, 7-4, 10-2
use during development 16-2
use to declare chain 10-4
use to declare instance 3-5, 10-1
ups depend command 2-7, 10-8, 22-79
reference 22-27
ups directory 3-5, 7-1, 10-2, 15-6, 27-11
description 15-6
locate using ups list -K 22-52
overwrite 10-14
test if needs update 10-13
UPS environment (See environment)
ups exist command 10-16, 22-33
reference 22-31
ups flavor command 2-1
-H option (specifies other flavor) 22-36
-l option (returns flavor table) 22-36
number options (specify OS level) 2-2
obtain flavor levels 2-2
obtain flavor table 2-2
reference 22-35
ups get command
reference 22-39
ups help command
reference 22-41
UPS initialization file 11-6
ups installasroot command 12-6
ups list command 2-2, 3-6, 10-3, 10-5
condensed output 2-3, 22-46
default output fields 22-45
for db managers and product installers 27-1
formatted output 2-3, 22-45
-K option
for script-readable format 2-3, 22-46
keyword arguments 22-46
use to locate product files 22-52
keywords for -K option 2-3
list all current products 2-4
list all output fields 2-6
long listing 22-51
parse output
in perl 22-52
in sh script 22-53
reference 22-43
ups modify command
editing database files 10-11
reference 22-55
UPS product overlay (See overlays)
UPS product requirements (See dependencies)
UPS products
accessibility 10-16
announcement policies 17-10
bin directory 15-6
build and distribute using template_product 18-1
catman directory 15-7
compilation options 1-4
definition 1-3
directory structure 15-6
distribution restrictions 20-11
distribution via CD-ROM 20-14
doc directory 15-7
documentation storage 15-7
examples directory 15-7
files and directories to include 19-1
hardcoded locations 15-3
html directory 15-7

include directory 15-7
Info directory 15-7
INSTALL_NOTE file 15-6
installation methods, summary 3-1
installed with different name than on server 3-8
interruption during installation 3-8
lib directory 15-7
list on distribution node 4-1
man directory 15-7
news directory 15-7
overlays 16-7
permissions set at installation 3-7
proprietary products
on fnkits 4-8
qualifiers 1-4
README file 15-6
special categories, flagging 20-12
src directory 15-7
support levels 17-10
test directory 15-7
third-party 15-3
toInfo directory 15-6
toman directory 15-6
ups directory 7-1, 15-6, 27-11
ups script 14-1
ups setup command (for troubleshooting) 9-1, 10-17
ups start command 14-2, 14-5
reference 22-59
usage in autostart 14-3
ups stop command 14-2
reference 22-63
usage in autostart 14-4
ups tailor command 3-9, 22-69
reference 22-67
ups touch command
reference 22-71
ups unconfigure command 10-7, 10-9, 22-79
reference 22-73
ups undeclare command
reference 22-77
remove chain 10-6, 22-77
remove product instance 10-7, 10-8, 22-79
syntax and common options
for chain removal 10-6
for product removal 10-8
-y and -Y options to remove root directory 10-8
ups verify command 10-10
reference 22-81
run by ups modify 10-11
use in troubleshooting problem installations 9-1, 10-17
ups.cgi script 20-2
description 20-5
UPS/UPD/UPP installation components 1-1
UPS_ARCHIVE_FILE keyword 20-9, 20-10
description 27-7
UPS_ARCHIVE_FILES keyword
use in updconfig 31-4
UPS_DB_VERSION keyword 30-2
description 27-7, 28-3, 29-2
UPS_DIR keyword 22-48, 27-11
as set by -U option 24-5
description 27-7, 28-3
UPS_EXTENDED variable 24-7
UPS_PROD_DIR keyword 3-4
description 27-7
use in updconfig 31-3
ups_shutdown script 14-1, 14-2, 14-5
ups_startup script 14-1, 14-2, 14-5
UPS_TABLE_DIR keyword
description 27-7
use in updconfig 31-3
UPS_TABLE_FILE keyword
description 27-8
use in updconfig 31-4
UPS_THIS_DB keyword
description 27-7
use in updconfig 31-3
UPS_UPS_DIR keyword
description 27-8
use in updconfig 31-3
upsdb_list file 12-2
for AFS 12-5
upsdb_list variable 13-3
ups-decl.cgi script 20-2, 20-11
access restrictions 20-6
description 20-5
user comment solicitation INT-5
USER keyword
description 27-8
user-defined chains 1-4
user-defined commands 33-3
user-defined keywords 27-2
US-only products 21-3

V

variables (read-only) defined within UPS 34-18
vendor-supplied products
rebuilding 16-6
verbose command output (-v) 9-1, 10-17
version files 1-6, 22-79
and product removal 10-7
creating 28-1
description 28-1
examples 28-3
information included in 28-1
information storage format 28-1
instance matching within 26-3
location 28-1
overview 27-1
table location specification in 28-5
VERSION keyword 22-48
description 27-8, 28-3, 29-2
version of product 1-3
version specification in commands 25-1

W

web server
access file 20-11
log file on fnkits 21-7
on distrib node 20-1
prerequisites for cgi scripts 20-7

writeCompileScript function
    description 34-16
www
    download products from 16-5